# VALORES NULOS
# en el diseño de
# Bases de Datos Relacionales

Universidad de Castilla la Mancha
Escuela Superior de Informática de Ciudad Real

BASES DE DATOS

Alumno: *Jose Gabín Camacho*
Profesor: *Francisco Ruiz González*

Fecha: Enero - 2000

# *Valores Nulos en el diseño de Bases de Datos Relacionales*

# *INDICE*

# *INDICE GENERAL*

# Introducción.

En el trabajo que presentamos a continuación vamos a hablar de un punto de suma importancia para que los diseños de bases de datos según el modelo relacional sean eficientes y de calidad, los Valores Nulos en el diseño de Bases de Datos Relacionales.

A lo largo de todo el trabajo nos centraremos en cada capítulo en cómo actúan los valores nulos sobre la información de nuestra base de datos.

El trabajo se ha organizado en seis capítulos. En el capítulo 1 comenzamos dando una pequeña introducción a las bases de datos, en este capítulo se definen los conceptos generales de base de datos, decimos también que es un sistema de gestión de bases de datos así como las ventajas y desventajas de implantar una base de datos. Un objetivo importante de este primer capítulo es fijar una terminología clara en la materia.

El capítulo 2 es una pequeña introducción al modelo relacional, donde se estudian los fundamentos de este modelo. Como sabemos, con las bases de datos intentamos guardar datos del mundo exterior de forma estructurada y, sobre todo, nuestro principal problema es captar la semántica de los datos. Con el concepto de semántica, en este entorno, nos referimos a todas las relaciones que puede haber entre los datos. Al modelizar una determinada zona cerrada del mundo exterior (universo de discurso) intentamos captar toda la información que es relevante para nosotros, en muchos casos esta información no son solo datos. Veremos una serie de características muy generales de cómo se enfoca la representación de datos del universo de discurso mediante el modelo relacional. Para ello veremos también las 12 reglas de Cood.

Seguiremos viendo en el capítulo 3 una definición conceptual de lo que son los valores nulos en el modelo relacional y su tratamiento e intentaremos dar respuesta en este apartado a las preguntas: ¿Qué es la Falta de Información? y ¿Qué son los Valores Nulos?.

Un valor nulo es un indicador que dice al usuario, que el dato falta o no es aplicable. Por conveniencia, un dato que falta normalmente se dice que tiene el valor NULO, pero el valor NULO no es un valor de dato real. En vez de ello es una señal o un recordatorio de que el valor falta o es desconocido.

Después de esta definición de valores nulos, veremos en el capítulo 4, cómo afectan éstos en la transformación desde el modelo Entidad/Relación al modelo Relacional. Las claves externas y las diferentes restricciones, de integridad y de usuario, serán explicadas de una forma práctica mediante ejemplos para una mayor comprensión del lector.

En el capítulo 5 hablaremos de la normalización en el diseño de bases de datos relacionales. La teoría de la normalización se desarrolló para que el diseño de bases de datos mejorara. Se exponen las distintas formas normales que existen y veremos las ventajas que proporciona el que una base de datos se encuentre en una u otra forma normal. En general, cuando un esquema relacional está en una forma normal superior a otra estamos eliminando redundancias y posibles inconsistencias, con lo que reducimos la posibilidad de errores, es por tanto, mejor cuanto más alta es la forma normal en la que se encuentra un esquema relacional. Al final de este capítulo se incluye una guía de cómo utilizar la teoría de la Normalización dentro del diseño lógico.

Y por último en el capítulo 6 trataremos de dar una visión práctica de cómo afectan los valores nulos en el manejo del lenguaje SQL (valores NULL).

Debido a la explosiva popularidad de SQL, este lenguaje es una de las tendencias más importantes de la industria informática actual. En los últimos años SQL se ha convertido en el lenguaje estándar de bases de datos. Alrededor de 100 productos de bases de datos ofrecen SQL, ejecutándose en un rango de sistemas informáticos que van desde las computadoras personales a los sistemas basados en una computadora central. Se ha adoptado y extendido un estándar internacional de carácter oficial de SQL. SQL juega un papel principal en la arquitectura de bases de datos de los principales proveedores de computadoras, y está en el núcleo de las estrategias de bases de datos de Microsoft. Desde sus oscuros comienzos como un proyecto de investigación de IBM, ha ganado importancia a pasos agigantados, tanto como una tecnología informática relevante, como una poderosa influencia de mercado. Veremos los distintos operadores y sus funciones, así como la funcionalidad nueva en el SQL2 de los operadores básicos del SQL1.

Veremos pues, como afecta la falta de datos, es decir los Valores Nulos, en el Diseño de Bases de Datos Relacionales a través del lenguaje SQL.

# 1. Introducción a las Bases de Datos.

*Un Sistema de Información (SI) es un conjunto de elementos ordenadamente relacionados entre sí de acuerdo a ciertas reglas, que aportan a la organización a la que sirven la información necesaria para el cumplimiento de sus fines. [MOTA94]*

*Una Base de Datos (BD) es una colección de datos estructurados según un modelo que refleje las relaciones y restricciones existentes en el mundo real. Los datos que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción han de ser únicas estando almacenadas junto a los mismos. Por último, los tratamientos que sufran estos datos tendrán que conservar la integridad y seguridad de éstos. [MOTA94]*

*Una Base de Datos es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información y no predicable en tiempo.*

*Un Sistema de Gestión de Bases de Datos (SGBD) es una herramienta software que proporciona una interfaz entre los datos almacenados y los programas de aplicación que acceden a éstos y que se caracteriza fundamentalmente por permitir una descripción centralizada de los datos y por la posibilidad de definir vistas parciales de los mismos para los diferentes usuarios. [MOTA94]*

*Un SGBD es un conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministran, tanto a usuarios como a programadores, analistas o administrador, los medios necesarios para describir y manipular los datos almacenados en la base de datos, manteniendo su integridad, confidencialidad y seguridad.*

Los sistemas relacionales de bases de datos forman (SGBDR), junto con la lógica de los negocios y las interfaces gráficas los tres pilares de los sistemas de información actuales, pero de estos tres elementos ninguno quizá como los SGBDR tienen más claras sus fronteras.

Los últimos avances tecnológicos han acercado al usuario (empresarial o doméstico), un potencial de información prácticamente ilimitado, y para manejar esta información cada vez se necesitan sistemas de gestión de aplicaciones más complejos. Empresas proveedoras tales como Informix, PostGres, Oracle y Microsoft han aportado grandes soluciones a estos temas.

Una de las tareas de los SGBDR, consiste en presentar a los programadores una visión lógica según un modelo aceptado independientemente de la organización física de los mismos. Esta independencia ha conseguido que la arquitectura física de los datos sea más optima para el sistema sobre el que va trabajar la base de datos.

- **Ventajas de las Bases de Datos.**

1. Independencia de datos y tratamiento. El cambio en los datos no implica el cambio en programas y viceversa, por lo tanto menor coste de mantenimiento.
2. Coherencia de resultados. Reduce la redundancia.
3. Mejora en la disponibilidad de los datos. Idea de catálogos.
4. Seguridad de los datos. Restricciones de seguridad sobre accesos y operaciones. No todos los usuarios de la BD pueden acceder a toda la importación y tampoco en el mismo modo.
5. Mayor eficiencia en la gestión de almacenamiento y efecto sinérgico.

- **Desventajas de las Bases de Datos.**

1. Situación sistema tradicional <> Situación sistema BD.
2. Fuerte coste inicial. Programa, personal y equipos.
3. Rentable a medio o largo plazo.
4. No hay standard.
5. No solo se puede cambiar datos sino también el enfoque del sistema.

*Una base de datos bien diseñada es una garantía del resultado de las aplicaciones, un aval de su futuro mantenimiento, y una tranquilidad de las posibles variaciones que puedan darse en los datos con las implicaciones que ello conlleva [LUCA93].*

# 2. El Modelo de Datos Relacional.

- **Introducción.**

*Un modelo de datos es un conjunto de conceptos, reglas y convenciones que nos permiten describir los datos del universo de discurso (UD), constituyendo una herramienta que facilita la interpretación de nuestro universo del discurso y su representación en forma de datos en nuestro sistema de información.*

El Modelo de Datos Relacional fue propuesto inicialmente por E. Cood en el año 1970 imponiéndose sobre los modelos previos (jerárquico y red) durante la década de los ochenta. Actualmente, es el modelo elegido para la construcción de casi todos los sistemas de gestión de bases de datos comerciales. El motivo de su éxito reside por un lado en su sencillez, el usuario percibe la base de datos como un conjunto de "tablas", y, por otro lado, en el carácter declarativo de sus lenguajes de consulta y manipulación.

El modelo relacional esta basado fundamentalmente en el concepto de *relación.* Una relación es una colección de tuplas, cada una de las cuales esta formada por una serie de atributos, estos atributos son los mismos en cada una de las tuplas.

Una relación se puede representar como una tabla, en la cual cada atributo etiqueta las columnas de dicha tabla y cada fila es una tupla. Debido a este modo de representación los términos relación y tabla se han convertido en sinónimos y a lo largo de este trabajo los usaremos indistintamente. También son sinónimos los términos atributo y campo.

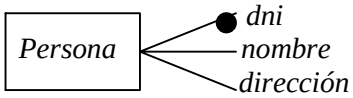
Estas tablas para poder ser consideradas relaciones deben cumplir además unas determinadas condiciones, llamadas restricciones, que veremos a continuación.

Las relaciones tienen un nombre. Para denotar una relación se suele usar la siguiente notación:

Nombre_relación (Atributo1, Atributo2, ...)

Si algún atributo se encuentra subrayado significa que forma parte de la clave principal y si alguno se encuentra en letra itálica es que se trata de una clave ajena. (Veremos estos conceptos más adelante).

***Ejemplo:*** *Para representar la entidad Persona del siguiente diagrama Entidad/Relación se define una relación con el mismo nombre.*




| *Definición de dominios* | *Dominios asociados* |
|---|---|
| *dominio_dni: entero* | *dominio_dni: entero* |
| *dominio_nombre: char(20)* | *dominio_nombre: char(20)* |
| *dominio_dirección: char(15)* | *dominio_dirección: char(15)* |

*La definición de la correspondiente relación es la siguiente:*

*Persona(dni: dominio_dni, nombre: dominio_nombre, dirección: dominio_dirección)*

- **Objetivos del modelo Relacional.**

El Objetivo fundamental del modelo relacional es "*mantener la independencia de la estructura lógica respecto al modo de almacenamiento y a otras características de tipo físico*". Hay también una serie de objetivos propuestos por Cood(1990) que se pueden resumir en los siguientes:

- **Independencia fisica.**
  El modo en que se almacenan los datos no influye en su manipulación lógica, y por tanto, no es necesario modificar los programas por cambios en el almacenamiento fisico. (Codd concede mucha importancia a este aspecto → Independencia **de ordenación**, independencia **de indexación** e independencia **en criterios de acceso**).

- **Independencia Lógica.**
  La insercción, modificación o eliminación de objetos en la base de datos no debe repercutir en los programas y/o usuarios que esten accediendo al subconjunto parcial de la base de datos (vistas).

- **Flexibilidad.**
  Poder presentar a cada usuario los datos de la forma que prefiera.

- **Uniformidad.**
  Las estructuras logicas de datos presentan un estado uniforme, lo que facilita la concepción y manipulación de la base de datos por parte de los usuarios.

- **Sencillez.**
  Las características anteriores, así como unos lenguajes de usuario muy sencillos, producen como resultado que el modelo de datos relacional sea fácil de comprender y de utilizar por parte del usuario final.

- **Dominio y Atributo.**

Un dominio D es el conjunto finito de valores homogéneos y atómicos, V1, V2, ..., Vn, caracterizados por un nombre; decimos valores homogéneos porque son todos del mismo tipo, y atómicos porque son indivisibles en los que al modelo se refiere, es decir, si se descompusiesen perderían la semántica a ellos asociada.

Todo dominio debe tener un nombre por el que referirnos a él y un tipo de datos.

Los dominios pueden definirse por intensión y por extensión. Un dominio es el conjunto del que un determinado atributo toma sus valores.

Un atributo A es el papel tiene un determinado dominio D en una relación; se dice que D es el dominio de A y se denota como dom(A).

El universo de discurso de una base de datos relacional está compuesto por un conjunto finito y no vacio de atributos, A1, A2, ..., An, estructurados en relaciones; cada atributo toma sus valores de un unico dominio (dominio subyacente) y varios atributos pueden tener el mismo dominio subyacente.

*El número de atributos que forman una relación es llamado* Grado *de esa relación y el número de tuplas que tiene una relación es llamado* Cardinalidad*, es decir el grado y la cardinalidad son el número de columnas y de filas de la tabla respectivamente.*

- **CLAVES: Concepto y tipos.**

  - **Clave Candidata.**
    Es un atributo o conjunto no vacio de atributos que identifican univoca y minimamente a una tupla, es decir no hay dos tuplas con dos claves candidatas iguales. Por supuesto siempre existirá una clave candidata y en una relación puede existir más de una.

  - **Clave primaria.**
    La que el usuario escoge de las claves candidatas y que será única. Sirve para referenciar las tuplas de la relación.
    *Ningún atributo que forme parte de la clave primaria puede tomar un valor nulo.*

  - **Claves alternativas.**
    Son aquellas claves candidatas que no han sido escogidas como clave primaria.

  - **Clave ajena.**
    Se denomina Clave Ajena de una relación R2 a un conjunto no vacio de atributos cuyos valores han de coincidir con los de la clave primaria de una relación R1 (R1 y R2 no son necesariamente distintas). La clave ajena y la clave primaria deben estar definidas sobre los mismos dominios.

- # Restricciones de Integridad en bases de datos Relacionales.

En el modelo relacional, al igual que en otros modelos, existen restricciones, es decir, estructuras u ocurrencias no permitidas, siendo preciso distinguir entre restricciones inherentes y restriccions de usuario. Los datos almacenados en la base de datos han de adaptarse a las estructuras impuestas por el modelo (por ejemplo, no tener tuplas duplicadas) y han de cumplir las restricciones de usuario para constituir una ocurrencia válida del esquema.

## a) Restricciones INHERENTES.

De la definición matemática de relación se deduce inmediatamente una serie de características propias de una relación que se han de cumplir obligatoriamente, por lo que se trata de restricciones inherentes y que, como ya hemos señalado, diferencian una relación de una tabla:

- No hay dos tuplas iguales.
- El orden de las tuplas no es significativo.
- El orden de los atributos no es significativo.
- Cada atributo solo puede tomar un valor del dominio, no adminitiendose por tanto dos grupos repetitivos.
- Se debe cumplir la regla de **Integridad de entidad** → *'Ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor desconocido o inexistente'*.
  Es decir, ningún valor de la clave primaria puede ser NULO. Esto es porque el valor de la clave primaria se usa para identificar las tuplas individuales de ina relación; permitir valores nulos para la clave primaria implica que no se pueda identificar algunas tuplas.

**Ejemplo.**

ESTUDIANTE

| NOMBRE | EDAD | SEXO |
|---|---|---|
| Juan Carlos | 19 | Varón |
| Jose Gabín | 24 | Varón |
| Susana Mera | 23 | Hembra |

CURSO

| CODIGO | INSTRUCTOR |
|---|---|
| CS347 | Miguel Angel |
| CS311 | Sandra López |

EXAMEN

| NUMERO_CURSO | NOMBRE_ESTUDIANTE | CALIFICACIÓN |
|---|---|---|
| CS347 | Juan Carlos | 5.65 |
| CS347 | Jose Gabín | 7.5 |
| CS311 | Susana Mera | 8.25 |

Si dos o mas valores tuvieran un valor nulo como clave primaria, no podríamos distinguirlas. En las tablas del ejemplo vemos como debido a las restricciones de integridad de las entidades, NOMBRE no puede ser nulo en ninguna tupla de ESTUDIANTE, CODIGO no puede ser nulo en ninguna tupla de CURSO, y el par (NUMERO_CURSO, NOMBRE_ESTUDIANTE) debe tener ambos valores no nulo en cualquier tupla de examen.

## b) Restricciones de USUARIO.

Podemos considerar la restricción de usuario, dentro del contexto relacional, como un predicado definido sobre un conjunto de atributos, de tuplas o de dominios, que debe ser verificado por los correspondientes objetos para que éstos constituyan una ocurrencia válida del esquema.

**Integridad referencial** → *Si la relación R2 tiene un descriptor que referencia a la clave primaria de la relación R1 todo valor de dicha clave (Clave ajena R2) debe concordar con los valores de la clave primaria R1 o ser NULO. R1 y R2 son claves necesariamente distintas.*

Además, la clave ajena puede formar parte de la relación R2.

Tambien se deben definir las acciones a tomar en caso de acciones de modificación y de borrado:

- **Operación restringida (RESTRICT).**
  Solo se puede borrar una fila de la tabla que tiene clave primaria referenciada si no existen filas con esa clave en la tabla referenciada.

- **Operación con transmisión en cascada (CASCADE).**
  El borrado o la modificación de una fila de la tabla que contiene la clave primaria lleva consigo la modificación de las tablas cuya clave ajena coincida con la clave primaria modificada.

- **Operación con puesta a NULOS (SET NULL).**
  El borrado o la modificación de una fila de la tabla que contiene la tabla primaria lleva consigo la puesta a nulos de los valores de la clave ajena de las filas de la tabla que referencia, cuya clave coincida con el valor de la clave primaria de la tabla referenciada.

- **Operación con puesta a valor por defecto (SET DEFAULT).**
  El borrado o modificación de filas de tuplas de una relación que contiene la clave primaria referenciada lleva consigo poner el valor por defecto a la clave ajena de la relación que refefencia; valor por defecto que habría sido definido al crear la tabla correspondiente.

- **Operación que desencadena un procedimiento de usuario.**
  El borrado o la modificación de tuplas de la tabla referenciada pone en marcha un procedimiento definido por el usuario.

**Ejemplo.**

En la base de datos del ejemplo anterior, la relación EXAMEN tiene como clave primaria (NUMERO_CURSO, NOMBRE_ESTUDIANTE). El atributo NOMBRE_ESTUDAINTE, se refiere al estudiante que hizo el exámen, por tanto, se puede designar NOMBRE_ESTUDIANTE como clave ajena de EXAMEN refiriendose a la relación ESTUDIANTE. Esto significa que un valor de NOMBRE_ESTUDIANTE en cualquier tupla t1 de la relación EXAMEN debe:

1) Coincidir con un valor de la clave primaria de ESTUDIANTE (el atributo NOMBRE) en alguna tubla t2, o bien,

2) Ser NULO. Sin embargo un valor nulo de NOMBRE_ESTUDIANTE no está permitido porque viola la restricción de integridad de entidades de la relación EXAMEN.

Supóngase que se tuviera una clave ajena denominada NOMBRE_EDIFICIO en examen (especificando el lugar del examen), que se refiera a la clave primaria de una relación denominada EDIFICIO; de nuevo habría una restricción referencial, pero ahora se permitirían valores nulos de NOMBRE_EDIFICIO en EXAMEN (especificando que el lugar del exámen se desconoce).

**c) Otras restricciones.**

- Definir predicados sobre:
  - Atributos de una relación.
  - Tuplas de una relación.
  - Atributos de varias relaciones.
  - Dominios.
- Evaluación de restricciones.
  - Con cada operación
  - Diferido.

## · Operadores.

Existen varios operadores asociados al modelo relacional y que se aplican a las relaciones, todos ellos conformal el álgebra relacional y los estudiaremos de una forma práctica mediante ejemplos en SQL en el capítulo 6, SQL: El lenguaje de las Bases de Datos Relacionales, señalando también que ocurre con los operadores y los valores nulos.

- **Reglas de Codd para SGBD Relacionales.**

Codd es el creador de las bases de datos relacionales. En el año 1985 en un artículo publicado por la revista Computerworld, Codd daba las siguientes reglas que debe cumplir cualquier base de datos que desee considerarse relacional:

0. Un SGBD-Relacional debe emplear para gestionar la BD exclusivamente sus facilidades relacionales.
1. *Regla de información:* Toda la imformación en la Base de datos es representada en una y sola una forma: valores en columnas de filas de tablas.
2. *Regla de acceso garantizado:* Cada valor escalar individual puede ser direccionado indicando los nombres de la tabla, columna y valor de la clave primaria de la fila correspondiente.

***3. Tratamiento sistemático de valores nulos:*** *El SGBD debe soportar la representación y manipulación de información desconocida y/o no aplicable. Los valores Nulos (distinto de la cadena de caracteres vacía o de una cadana de caracteres blancos, y distinta del cero o de cualquier otro número) se soportan en los SGBD completamente relacionales para representar la falta de información inaplicable, de un modo sistemático e independiente del tipo de datos*

4. *Debe tener un catálogo en línea dinámico* (diccionario de datos) *basado en el modelo relacional.*
5. *Sublenguaje completo de datos*: El SGBD debe soportar al menos un lenguaje relacional:
   a) Con sistanxis lineal bien definida.
   b) Que pueda ser usado interactivamente o en programas (embebido).
   c) Con soporte para operaciones de:
      - definición de datos.
      - definición de vistas.
      - manipulación de datos (p.e. recuperación y modificación de tuplas).
      - restricciones de seguridad e integridad.
      - autorizaciones.
      - gestión de transacciones.
6. *Modificación de vistas:* todas las vistas teóricamente modificables deben poder serlo en la práctica.
7. *Inserción, modificación y borrado de tuplas de alto nivel.*
8. *Independencia física de los datos.*
9. *Independencia lógica de los datos.*
10. *Independencia de integridad:* Las restricciones de integridad deben estar separadas de los programas, tal que se puedan modificar éstas sin afectar a aquellos.
11. *Independencia de distribución:* Las aplicaciones no deben verse afectadas al distribuir (dividir entre varias máquinas), o al cambiar la distribución ya existente de lña Base de Datos.
12. *Regla de no subversión:* Si el sistema posee una interface de bajo nivel (p.e. llamadas en C), éste no puede subvertir el sistema; por ejemplo, pudiendo evitar restricciones de seguridad o integridad.

# 3. Valores Nulos en el modelo Relacional.

- **Concepto de Valor Nulo.**

Si bien los valores nulos no son un concepto exclusivo del modelo relacional, ha sido en el contexto de este modelo donde se han abordado su estudio de manera más sistemática y donde se están realizando más investigaciones a fin de formalizar su tratamiento, permitiendo así resolver los numerosos problemas teóricos y prácticos que rodean este tema.

*Se puede definir el **valor nulo** o valor ausente como una **marca** utilizada para representar información desconocida, inaplicable, etc. [DEMI93].*

La definición de tupla que se ha dado (todo atributo tiene un valor asociado) no permite contemplar el caso en el que se desconoce el valor de un atributo en una tupla, situación que es frecuente y que se había considerado en el modelo Entidad/Relación. Por ello, se va a introducir la hipótesis de la existencia de un ***valor nulo*** en todo dominio del esquema. ***Este valor nulo denota la ausencia de información, es decir el desconocimiento de la propiedad representada por el atributo para el objeto representado por la tupla;*** cuando esto sucede se dice que dicho atributo tiene valor nulo en esa tupla. Es importante destacar que el valor nulo aún siendo un valor de cualquier dominio tiene un comportamiento distinto frente a los operadores.

*Un **valor nulo** es un indicador que dice al usuario, que el dato falta o no es aplicable. Por conveniencia, un dato que falta normalmente se dice que tiene el valor NULO, pero el valor NULO no es un valor de dato real. En vez de ello es una señal o un recordatorio de que el valor falta o es desconocido. [GROFF98]*

La necesidad de los valores nulos es evidente por diversas razones como pueden ser:

- Existencia de tuplas con ciertos atributos desconocidos en ese momento.
- Necesidad de añadir un nuevo atributo a una tabla ya existente; atributo que en el momento de introducirse no tendrá ningún valor para las tuplas de las relaciones.
- Posibilidad de atributos inaplicables a ciertas tuplas.

***Ejemplo.***

| *DNI* | *NOMBRE* | *DIRECCIÓN* |
|---|---|---|
| *35784843* | *María Gutiérrez* | *?* |

*La siguiente tupla:*

***{(dni, 35784843), (nombre, María Gutiérrez), (dirección, ?)}***

*que se puede denotar como tupla T, es una representación de esta función mediante pares (atributo, valor). En esta tupla es atributo dirección tiene valor nulo indicando que se desconoce la dirección de esta persona.*

El permitir que algunos atributos puedan tomar valores nulos tiene efectos importantes en las definiciones dadas anteriormente en el capítulo 2, Modelo Relacional, que deben ser revisadas y adaptadas. Vamos a tratar de exponer esas diferencias.

*Representaremos los valores nulos con el signo ?.*

**a) Dominio.**

El concepto de dominio para los atributos que puedan tomar valor nulo cambia significativamente, ya que incluiremos en su conjunto de valores el signo ? como un elemento más, distinto a todos los demás.

**b) Relación.**

Una relación es un conjunto de tuplas. Por tanto, no puede haber tuplas duplicadas. Dadas dos tuplas, consideraremos que una es duplicada de la otra cuando todos los atributos que son nulos en una lo son también en la otra, y todos los que no son nulos tienen en la otra el mismo valor.

**c) Superclave.**

Diremos que un atributo A es una superclave cuando sus valores no nulos son todos distintos. Si A es compuesto, interpretaremos que no es nulo cuando todos sus componentes lo sean.

- **Falta de Información.**

*El problema de la información que falta tiene gran importancia: en el mundo real la información a menudo es incompleta, por lo cual es preciso contar con una forma de manejar estas situaciones en nuestros sistemas formales de bases de datos. Sin embargo, pensamos que los* ***VALORES NULOS*** *al estilo SQL no resuelven ese problema. De hecho pensamos que todavía no se comprende del todo el problema de la información que falta y que aún se desconoce una solución completamente satisfactoria.*

*No obstante, debemos recalcar que algunos otros autores opinan lo contrario, y se han presentado varias propuestas par manejar la información que falta. Quizá la más conocida es la de Codd, quien considera ahora su estrategia como parte integral del modelo relacional [DATE93].*

El enfoque de Codd se basa en la lógica de tres valores. La idea fundamental es que cualquier posición de fila y columna en la base de datos se podría marcar como nula (siempre y cuando no lo impidan restricciones de integridad), donde "nula" significa que falta el elemento de información correspondiente por desconocerse su valor.

Supongamos ahora que dos de estas posiciones, digamos x e y, se marcan como nulas, y consideremos la pregunta "¿Son estos dos nulos iguales entre si?". La respuesta a esta pregunta no puede ser "Si", porque eso implicaría que los nulos ya no son desconocidos (del todo); se sabría que uno es igual al otro. Por lo mismo la respuesta tampoco puede ser "No", porque eso implicaría que uno es diferente del otro. En vez de ello, la respuesta debe ser "no sabemos".

En términos más generales, el resultado de evaluar la comparación ***x theta y*** (donde theta es cualquier operador de comparación escalar simple) no es ***verdadero ni falso, sino desconocido,*** si x es nulo o y es nulo o los dos lo son. De ahí la lógica de tres valores de Cood; verdadero, falso y desconocido son los tres "valores de verdad" de esa lógica trivaluada.

Ahora definiremos versiones para la lógica de tres valores de los operadores lógicos NO(NOT), Y(AND) y O(OR) mediante las siguientes tablas de verdad:

Tabla de verdad de AND.

| **Y** | **VERDADERO** | **FALSO** | **DESCONOCIDO** |
|---|---|---|---|
| VERDADERO | VERDADERO | FALSO | DESCONOCIDO |
| FALSO | FALSO | FALSO | FALSO |
| DESCONOCIDO | DESCONOCIDO | FALSO | DESCONOCIDO |

Tabla de verdad de OR.

| **O** | **VERDADERO** | **FALSO** | **DESCONOCIDO** |
|---|---|---|---|
| VERDADERO | VERDADERO | VERDADERO | VERDADERO |
| FALSO | VERDADERO | FALSO | DESCONOCIDO |
| DESCONOCIDO | VERDADERO | DESCONOCIDO | DESCONOCIDO |

Tabla de verdad de NOT.

| **NO** | **VERDADERO** | **FALSO** | **DESCONOCIDO** |
|---|---|---|---|
| | FALSO | VERDADERO | DESCONOCIDO |

También necesitamos un nuevo operador lógico, QUIZÁ. Por definición, si se aplica el operador QUIZÁ (MAYBE) a una expresión lógica p, el resultado será verdadero si al evaluar p se obtiene desconocido, y falso en cualquier otro caso. Así pues, QUIZÁ se define con la siguiente tabla de verdad:

Tabla de verdad de MAYBE.

| QUIZÁ | VERDADERO | FALSO | DESCONOCIDO |
|---|---|---|---|
| | FALSO | FALSO | VERDADERO |

Existen una serie de problemas que resultan del esquema anterior y que veremos a continuación:

1. **Problemas psicológicos.**

La lógica de tres valores está llena de trampas para los incautos. Un ejemplo sencillo: las consultas "obtener los proveedores de una determinada ciudad", por ejemplo "Londres" y "obtener los proveedores que no están en esa ciudad" juntas no producirán entre las dos todos los proveedores (es necesario incluir también la consulta "obtener los proveedores que quizá estén en Londres").

Como ejemplo un poco más problemático, supongamos una ligera variación en una base de datos que contiene información sobre proveedores y piezas que estos suministran, en la cual la tabla SP tiene otra columna, NUMEMB, la cual actúa como clave primaria, y sí se permiten nulos en las columnas SP.S# y SP.P#; consideremos la consulta

*SX.SNOMBRE WHERE NOT EXISTS SPX (SPX.S# = SX.S# AND SPX.P# = 'P2')*

¿Qué significa esta columna? Entre las posibles interpretaciones se encuentran por lo menos las siguientes:

a) Obtener los nombres de los proveedores que no suministran la pieza P2.
b) Obtener los nombres de los proveedores de los cuales no se sabe si suministran o no la pieza P2.
c) Obtener los nombres de los proveedores de los cuales se sabe que no suministran la pieza P2.
d) Obtener los nombres de los proveedores de los cuales o bien se sabe que no suministran la pieza P2 o no se sabe si lo hacen o no.

2. **Definición de duplicados.**

Si x es nula, entonces x no es igual a sí misma, porque el resultado de evaluar la comparación x = x es desconocido, no verdadero. Sin embargo, Codd requiere que de todos modos que x se considere como duplicado de sí misma, a fin de que la eliminación de duplicados funcione en forma "correcta" cuando se obtiene (digamos) una proyección. Así, la pregunta "¿Dos nulos son iguales?" debe interpretarse de acuerdo con el contexto.

En el estándar SQL, la respuesta es "Si" en el caso de DISTINCT, ORDER BY y GROUP BY, y es "No" en el caso de WHERE y HAVING.

3. **Anomalías lógicas.**

El desarrollo anterior hace surgir una gran número de anomalías lógicas. A continuación presentamos una lista de todas ellas:

- Si x es una variable escalar, no se garantiza que el resultado de evaluar la comparación x = x sea verdadero.

- Si x es una variable numérica, no se garantiza que el resultado de evaluar la expresión x – x sea cero.
- Si X es un conjunto, no se garantiza que el resultado de evaluar la comparación X subconjunto de X sea verdadero.
- La intersección ya no es un caso especial de la reunión natural.
- No se garantiza que la reunión natural de una relación consigo misma sea igual a la relación original.
- Si p es una expresión lógica, no se garantiza que el resultado de evaluar p OR NOT p sea verdadero, ni que el resultado de evaluar p AND NOT p sea falso.

Una consecuencia de lo anterior es la probabilidad e errores por parte tanto de los usuarios como de los optimizadores del sistema, por la aplicación de transformaciones simbólicas que son correctas en la lógica ordinaria de dos valores pero que pudieran no serlo en la lógica de tres valores. De hecho, se sabe que algunos sistemas disponibles en el mercado adolecen de errores en esta área.

## 4. Otros tipos de Nulos.

Hasta ahora nos hemos limitado a examinar un solo tipo de nulo, a saber, “valor desconocido”. Sin embargo, pueden existir varios otros tipos; por ejemplo, “la propiedad no es aplicable”, “el valor no existe”, “el valor no está definido”, ...; así Cood propone expandir el enfoque de la lógica de tres valores a una lógica de cuatro valores para manejar un tipo adicional de nulo, a saber “la propiedad no es aplicable”. Así pues, pareciera que n tipos de nulo conducirán a una lógica de (n+2) valores. Considerando todos los problemas que surgen en el caso simple de n = 1, seguramente deberá cuestionarse la conveniencia de aplicar el tratamiento a un n > 1 arbitrario.

En vista de la existencia de otros tipos de nulos, otro de los problemas que surgen es el de la **interpretación**. Por ejemplo, ¿Qué significan los valores nulos generados por una recuperación de reunión externa?. Los ejemplos sugieren significados diferentes en los distintos contextos.

En cuanto a las operaciones aritméticas con valores nulos, se toma como nulo el resultado de suma, multiplicar, restar o dividir un valor nulo con cualquier otro valor.

Por lo que respecta a los operadores algebraicos, cabe destacar la importancia de considerar los valores nulos para evitar pérdidas de información a la hora de realizar consultas a la base de datos, para lo cual aparecen nuevos operadores como el de combinación externa (outer join).

Otro aspecto que hay que tener en cuenta es la realización de operaciones de agregación con el fin de aplicar funciones estadísticas, como frecuencias, varianza, media, etc. En estos casos hay que establecer muy claramente si se han de tener o no en cuenta los valores nulos al efectuar los cálculos, o nos podremos llevar sorpresas como que la media de los valores de un atributo no sea igual a la suma de los valores dividido por el número de elementos.

El gran problema de los valores nulos es que ningún producto relacional hasta la fecha los ha instrumentado de manera coherente y satisfactoria; por ejemplo, hay veces que no se tienen en cuenta los valores nulos, como para las operaciones de agrupamiento, pero en otros casos sí, como en la ordenación o proyección.

- **Operadores relacionales con Valores Nulos.**

Existen varios operadores asociados al modelo relacional, *(estudiaremos éstos de una forma práctica mediante ejemplos en SQL en el capítulo 6, SQL: El lenguaje de las Bases de Datos Relacionales, señalando también que ocurre con los operadores y los valores nulos).*

El tratamiento de valores nulos exige definir operaciones y funciones específicas para valores nulos, entre otras:

- Operaciones aritméticas (+, -, *, :)
- Operaciones de comparación (=, <>, >, >=, <, <=)
- Tablas de verdad con lógica trivaluada o cuatrivaluada para los operadores NO(NOT), Y (AND), O (OR)
- Operaciones del álgebra relacional (unión, intersección, combinación).
- Funciones estadísticas (media, varianza, ...)

Entre los operadores especiales para el tratamiento de valores nulos, destaca el de combinación externa (outer join), que impide que se pierdan tuplas que desaparecen cuando se aplica la combinación interna.

Para evitar que las tuplas de una relación que no casan con ninguna tupla de la otra desaparezcan en el resultado podemos aplicar el operador de combinación externa que concatena esas tuplas con tuplas nulas. Cuando sea necesario crear tuplas nulas en la segunda tabla se utiliza el operador de combinación externa *"left outer join"* , mientras que si se crean las tuplas nulas en la primera tabla habrá que utilizar el operador de combinación externa *"right outer join"*. Puede también ser necesario crear tuplas nulas en ambas relaciones, denominándose en este caso el operador como combinación externa simétrica.

En SQL2 se admiten los tres tipos de operadores de combinación externa, pero en los productos comerciales sólo se admite utilizar en una misma consulta el derecho o el izquierdo, siendo habitual representarlo por (+).

Otro tipo de operaciones que tienen en cuenta los valores nulos, son los denominados *maybe (posible),* que, utilizando la lógica trivaluada admiten como tuplas resultantes no sólo las que cumplen el valor cierto, sino también las que cumplen *puede ser cierto*.

En el siguiente ejemplo se representa el caso de una combinación por igualdad de las relaciones R1 y r2, resultando la relación R3. Ahora bien aplicando la *combinación posible (maybe join),* el número de tuplas resultantes es considerablemente superior, ya que la primera tupla de la relación A < 1, 5 > no sólo se combina con la tupla < 5, 3> de la relación R2, sino que también se combina con las tuplas < ?, 6 > y < ?, 7 > de dicha relación.

| A | B |
|---|---|
| 1 | 5 |
| 1 | 7 |
| 2 | 4 |
| ? | 2 |
| 3 | 7 |

| A | B |
|---|---|
| 5 | 3 |
| 2 | 4 |
| ? | 5 |
| ? | 5 |

Resultado1: R3 = R1 $\theta$ R2
B=C

| A | B | C | D |
|---|---|---|---|
| 1 | 5 | 5 | 3 |
| ? | 2 | 2 | 4 |

Resultado2: R4 = R1 $\theta$ R2
B=C

| A | B | C | D |
|---|---|---|---|
| 1 | 5 | 5 | 3 |
| 1 | 5 | ? | 6 |
| 1 | 5 | ? | 5 |
| 1 | ? | 5 | 3 |
| 1 | ? | 5 | 4 |
| 1 | ? | ? | 6 |
| 1 | ? | ? | 5 |
| 2 | 4 | ? | 6 |
| 2 | 4 | ? | 5 |
| ? | 2 | 2 | 4 |
| ? | 2 | ? | 6 |
| ? | 2 | ? | 5 |
| 3 | ? | 5 | 3 |
| 3 | ? | 2 | 4 |
| 3 | ? | ? | 6 |
| 3 | ? | ? | 5 |

- **Valores por Defecto en lugar de Valores Nulos.**

Hay algunos autores como Date (1990), que propugnan evitar los *valores nulos*, mediante la utilización de *valores por defecto*. Pero este enfoque ha sido criticado por Cood (1990), ya que:

- No resuelve el problema del tratamiento de valores nulos, sino que simplemente proporciona un medio para representar información desconocida.

- Al representarse mediante un valor (y no un concepto especial), fuerza a comprobar las dependencias funcionales y otro tipo de dependencias al introducir la información, haciéndolo por tanto a destiempo.

- La representación del hecho de que un valor es desconocido no sólo depende del tipo de datos de cada columna, sino que también puede variar entre columnas con un mismo tipo de datos.

- Las distintas técnicas para tratar estos valores por defecto quedan embebidas en los programas, con pocas posibilidades de ser uniformes, sistemáticas y estar bien documentadas.

- Cada valor desconocido se trata, en definitiva, como si fuera cualquier otro valor.

- Representa un retroceso hacia enfoques no sistemáticos y *ad hoc* propios de la era prerrelacional.

- **Lógica Cuatrivaluada.**

La lógica cuatrivaluada surge de la necesidad de diferenciar dos tipos importantes, aunque hay más, de valores nulos: los inaplicables, esto es, que no tienen sentido, y los valores desconocidos aplicables, es decir, que momentáneamente son desconocidos pero deberían existir y puede que, en un determinado momento, lleguen a conocerse.

Esta diferencia, que puede parecer académica en exceso, es, de hecho, muy importante para reflejar con más precisión la semántica del universo del discurso a tratar y para responder de manera más inteligente a las consultas que se realicen sobre la base de datos.

Cood (1990) propone abandonar el término *valor nulo* para sustituirlo por el de *marca*, ya que:

- Los SGBDR no deberían tratar las marcas como si fueran cualquier otro valor.
- Al introducir la lógica cuatrivaluada se desdobla el concepto.
- Algunos lenguajes anfitriones tratan objetos que denominan nulos pero con diferente significado al de las marcas.
- Es más fácil decir marcado o sin marcar, que anulado o nulificado.

En la lógica cuatrivaluada se distingue, por tanto, entre un valor nulo o marca que representa información desconocida pero aplicable, que denominaremos A, de otro que representa información inaplicable, que denominaremos I.

En este caso, las operaciones aritméticas quedan modificadas como sigue, siendo & (suma, resta, división o multiplicación) y X un valor no nulo de la base de datos:

X & A = A & X = A
X & I = I & X = I
A & I = I & A = I
A & A = A
I & I = I

En las siguientes tablas se muestran las tablas de verdad para la lógica cuatrivaluada según Cood.

Tabla de verdad de AND

| Y | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| **CORRECTO** | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
| **APLICABLE** | APLICABLE | APLICABLE | FALSO | INAPLICABLE |
| **FALSO** | FALSO | FALSO | FALSO | FALSO |
| **INAPLICABLE** | INAPLICABLE | INAPLICABLE | FALSO | INAPLICABLE |

Tabla de verdad de OR

| O | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| **CORRECTO** | CORRECTO | CORRECTO | CORRECTO | CORRECTO |
| **APLICABLE** | CORRECTO | APLICABLE | APLICABLE | APLICABLE |
| **FALSO** | CORRECTO | APLICABLE | FALSO | FALSO |
| **INAPLICABLE** | CORRECTO | APLICABLE | FALSO | INAPLICABLE |

Tabla de verdad de NOT

| NO | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| | FALSO | CORRECTO | CORRECTO | INAPLICABLE |

**Ejemplo:** Aplicar estas tablas de verdad a la siguiente consulta sobre la tabla DOCUMENTOS que almacena todos los documentos de la biblioteca, sean artículos o libros:

*(idioma = "inglés") OR (editorial = "rama")*

a) Si se trata de un artículo en italiano, el primer término se evaluaría como *falso*, mientras que el segundo sería *inaplicable* (al carecer de sentido que un artículo lo publique una editorial), siendo el resultado *falso* tanto para Cood como para Gessert, ya que ambos utilizan la misma tabla de verdad para la operación booleana OR.
b) Si se trata de un libro en italiano cuya editorial desconocemos, daría resultado *aplicable* al ser el primer término *falso* y el segundo *aplicable*.

Sin embargo, para la consulta: *(idioma = "inglés") AND (editorial = "rama")*

a) Si se trata de un artículo en italiano, el primer término se evaluaría como *falso*, mientras que el segundo sería *inaplicable*, y el resultado sería *falso* para Cood e *inaplicable* para Gessert, ya que para Gessert bastará con que uno de los términos de una conjunción sea inaplicable para que su resultado lo sea.
b) Si se trata de un libro en italiano, daría como resultado *falso* para ambos.

Existe sin embargo cierto desacuerdo en torno al tema, puesto que hay autores, como Gessert (1990), que proponen ligeras variaciones, considerando que el valor inaplicable es el menos verdadero.

Además, la diferencia de Gessert con respecto a Cood es que el primero introduce un nuevo operador de negación denominado INV. Aplicando este operador, el *inverso* es *cierto* sería el valor *inaplicable* y viceversa, mientras que el *inverso* de *aplicable* resulta ser el valor *falso*, y viceversa. Por lo tanto las tablas de verdad quedan de la siguiente forma:

Tabla de verdad de AND

| Y | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| **CORRECTO** | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
| **APLICABLE** | APLICABLE | APLICABLE | FALSO | INAPLICABLE |
| **FALSO** | FALSO | FALSO | FALSO | INAPLICABLE |
| **INAPLICABLE** | INAPLICABLE | INAPLICABLE | INAPLICABLE | INAPLICABLE |

Tabla de verdad de OR

| O | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| **CORRECTO** | CORRECTO | CORRECTO | CORRECTO | CORRECTO |
| **APLICABLE** | CORRECTO | APLICABLE | APLICABLE | APLICABLE |
| **FALSO** | CORRECTO | APLICABLE | FALSO | FALSO |
| **INAPLICABLE** | CORRECTO | APLICABLE | FALSO | INAPLICABLE |

Tabla de verdad de NOT

| NO | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| | FALSO | CORRECTO | CORRECTO | INAPLICABLE |

Tabla de verdad de INV

| INV | CORRECTO | APLICABLE | FALSO | INAPLICABLE |
|---|---|---|---|---|
| | INAPLICABLE | FALSO | APLICABLE | CORRECTO |

*En conclusión podemos decir que en tanto no se terminen de manera satisfactorias estas investigaciones, será preferible en la práctica una estrategia basada en valores por omisión definidos por el administrador de bases de datos [DATE93].*

# 4. Transformación del modelo E/R al modelo Relacional.

Existe una forma habitual de representar la realidad usando el modelo relacional, como se puede observar cuando se traducen diagramas Entidad-Relación a esquemas relacionales. Los diagramas E/R vienen definidos fundamentalmente por entidades y relaciones. A continuación se indica como se representa en el modelo relacional cada uno de estos conceptos:

**a) Entidad.**
Como se ha visto anteriormente, una entidad se representa por una relación definida por el conjunto de la entidad y los dominios asociados a estos atibutos. Cada tupla de esta relación corresponderá a una ocurrencia de la entidad.

**b) Relación.**
Una relación R en la que forman parte las entidades E1, E2, ..., En, se representa por una relación definida por el conjunto de los atributos identificadores de las n entidades y los dominios correspondientes a dichos atributos (renombrando atributos si fuera necesario).

## • Reglas de transformación del Modelo Básico.

Para transformar un esquema E/R a un esquema Relacional se basa en tres principios:

1) Todo tipo de entidad se transforma en una relación.
2) Todo tipo de interrrelación N:M se transforma en una relación.
3) Todo tipo de interrrelación del 1:N se transforma en una nueva relación o se traduce en el fenómeno de propagación de clave.

NOTA: Al pasar se pierde semántica, ya que tanto entidades como interrelaciones se transforman en relaciones y también desaparece el nombre de algunas interrelaciones 1:N (al propagar la clave). Pero esta pérdida de semántica no afecta a la integridad de la base de datos, pues se añaden restricciones de integridad referencial.

### 1. Transformación de Entidades.

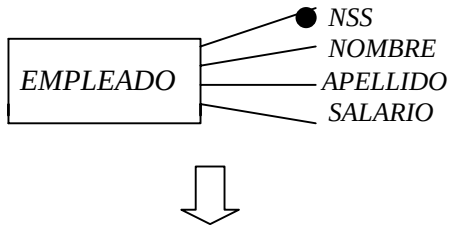
- **Transformación de Dominios.**
  Hay sistemas relacionales que no lo permiten. En el Modelo Relacional un dominio es un objeto que se definirá con el DDL, y la tranformación es directa mediante la sentencia de CREATE DOMAIN en SQL.

- **Transformación de Entidades.**
  Toda entidad se transforma en una relación de igual nombre mediante la sentencia CREATE TABLE en SQL.

- **Transformación de Atributos de Entidades.**
  Cada atributo de una entidad se transforma en un atributo de la relación (columna de la tabla), según tres subreglas:

- Atributos identificadores principales (AIP): pasan a ser la clave primaria de la relación y en SQL se utiliza la cláusula PRIMARY KEY.
- Atributos identificadores alternativos: se tranforman en claves alternativas, en SQL se utiliza la cláusula UNIQUE.
- Atributos no identificadores: serán columnas normales que podrán tomar valor nulo salvo que se indique lo contrario.

**Ejemplo:**




*EMPLEADO( NSS, NOMBRE, APELLIDO, SALARIO)*

2. **Transformación de Interrelaciones.** Según tipo:

- **Interrelación N:M.**

Se transforma en una nueva relación que tendrá como clave principal la concatenación de los AIP de los tipos de entidades participantes.

- Cada atributo de la clave primaria será clave externa respecto de una de las relaciones que representan las entidades participantes (Cláusula FOREING KEY).
- Además se deberá estudiar que ocurre con los modos de borrado y modificación de la clave primaria referenciada (Opci´pn restringida y SET NULL, SET DAFAULT o CASCADE en SQL).
- Cardinalidades máxima y mínima: se representan especificando rstricciones o aserciones (cláusula CREATE ASERTION)

Sintaxix: CLAVE AJENA (<Nombre columna>)
REFERENCIA TABLA (<Nombre tabla>)

Ejemplo:

*TABLA ESCRIBE (Cod_Libro, Nombre)*
*CLAVE PRIMARIA (Cod_Libro, Nombre)*
*CLAVE AJENA (Cod_Libro)*
*REFERENCIA TABLA LIBRO*
*EN BORRADO : CASCADA*
*EN ACTUALIZACION : CASCADA*
*CLAVE AJENA (Nombre)*
*REFERENCIA TABLA AUTOR*
*EN BORRADO : NULOS*
*EN ACTUALIZACION : CASCADA*

- **Interrelación 1:N entre dos entidades E1 y E2.** Hay pos posibles soluciones:

  a) **Propagación.**
  Propaga el atributo de identificación principal (AIP) del tipo de entidad con cardinalidad máxima 1 (E1) al tipo de cardinalidad máxima N (E2), desapareciendo el nombre de la interrelación y con la consiguiente pérdida de semántica. Dicho AIP será clave externa en la entidad E2 referenciando a E1.

  **b) Considerarla como una relación N:M.**
  Esta opción es la mejor en caso de:
  - Cuando el número de ocurrencias de E1 es muy pequeño y aplicando la solución a) podrían producirse muchos valores nulos.
  - Cuando se prevé que se convierta en N:M e el futuro.
  - Cuando la interrelación tiene atributos propios.

  Perdida de semántica en cardinalidades:

  - Cardinalidad máxima (Predicados) 1:N. Máximo es conocido (en tablas se permite limitar el número de filas).
  - Cardinalidad mínima. Si e1 tienen cardinalidades (0,1) se deben admitir valores nulos en la clave externa, y si es (1,1) no se deben admitir (cláusula NOT NULL en SQL).

- **Interrelaciones 1:1.**

  - Caso particular de las relaciones 1:N y M:N.
  - No existe una regla fija para su transformación, pudiendose crear una tabla nueva o propagar la clave. En este último caso, la propagación de la clave puede realizarse en ambas direcciones.
  - Criterios para la selección de una solución:
    - Cardinalidad mínima.
    - Semantica.
    - Eficiencia (Valores Nulos o no Nulos / Accesos más frecuentes).

  Hay varios casos que veremos a continuación:

  1. Si la cardinalidad es (0,1) en ambas entidades la interrelación se transforma en una nueva relación.
  2. Si la cardinalidad de una de ellas es (0,1) y la otra es (1,1) conviene propagar la clave de la entidad con cardinalidad (1,1) a la relación resultante de la entidad con cardinalidad (0,1)
  3. Si en ambas la cardinalidad es (1,1) se puede propagar clave en ambos sentidos o incluso propagar ambas claves.

**Ejemplo1:** Solución en el modelo Relacional.

*EMPLEADO (Cod_emp, .....)*

*DEPARTAMENTO (Cod_Dep, ....., Cod_emp)*

**Ejemplo 2.**

*MATRIMONIO (Cod_H, Cod_ M)*

*HOMBRE (Cod_H, .....)*
*MUJER (Cod_M, .....)*

- **Interrelaciones n-arias.**

  Cuando se tienen más de dos tipos de entidades participantes se aplica en general la misma regla que en las interrelaciones N:M, pudiendodes realizar simplificaciones según los casos. En general una interrelación con en participantes se transformará en n+1 relaciones.

- **Transformación de los atributos de interrelaciones.**

  a) Si la interrelación se transforma en una relación, todos sus atributos pasan a ser columnas de la relación.

  b) Si se transforma mesiante propagación de clave, sus atributos migran junto con la clave de la relación correspondiente, aunque suele ser mejor crear una nueva tabla para realizar esta migración de la interrelación con sus atributos.

- **Transformación de restricciones de usuario.**

  Se añaden como predicados al definir las tablas. Hay varios tipos :

  a) Comprobación del rango de valores de los atributos (dominios) mediante la cláusula RANGE BETWEEN en SQL ó cláusula IN al determinar por enumeración..

  b) Especificar todos los valores posibles de un atributo.

  c) Comprobar un predicado (interrelación sobre atributos), con la cláusula CHECK en SQL.

3. **Representación de tipos y subtipos en las Generalizaciones.**

a) Englobar todos los atributos del tipo y subtipos en una sola relación. Se utiliza cuando hay muy pocas diferencias entre los atributos de los subtipos.

b) Crear una relación para el tipo genérico y una más por cada subtipo. Cada subtipo tendrá como clave primaria el AIP del tipo, que a la vez será clave externa referenciando a la relación que representa a la entidad genérica. Es la mejor opción en la mayoría de los casos ya que conserva la semántica, aunque puede ser poco eficiente.

c) Crear una relación para cada uno de los subtipos, que contendrán todas ellas los atributos del tipo genérico.

Cuando el recubrimiento es Parcial el atributo discriminate admite valores nulos, NOT NULL en SQL.
Si hay solapamiento, el atributo discriminante es multivaluado, y como el modelo relacional esto no lo permite, habrá que crear una nueva relación aparte para contenerlo y asociarlo con una clave externa con la relación resultante del supertipo.

- **Dimensión temporal.**

  a) Si el tiempo es un tipo de entidad se transforma en una relación sin más.
  b) Si se contempla como atributos de interrelación de tipo FECHA, se convierten en atributos de la relación correspondiente, pero la clave primaria se puede ver afectada, teniendo que incluir algún atributo temporal.

- **Atributos derivados.**

  No existe una representación específica y puede haber varios casos:

  a) Se tratan como atributos normales. Se construye un procedimiento de usuario que calcule el valor cada vez que se inserten, borren o modifiquen ocurrencias que incluyen atributos que intervienen en el cálculo.
  b) No se añaden como atributos, creando procedimientos disparadores que calculan el valor cada vez que se recupera.

# 5. Diseño de Bases de Datos Relacionales: Normalización.

En este apartado se va a presentar la teoría de la normalización. Esta teoría proporciona un conjunto de conceptos que permiten conocer el grado de corrección de un esquema de base de datos relacional. Así, la teoría de la normalización se aplicará al esquema relacional obtenido en esta primera etapa del diseño lógico para refinarlo y comprobar que no existe ningún problema.

El problema del diseño lógico de bases de datos relacionales se puede expresar de la siguiente forma: *"Dado un conjunto de información, hay que decidir que relaciones existirán en la base de datos, y qué atributos contendrá cada una"*. Estudiaremos como realizar un buen diseño que evite los posibles problemas que se podrían originar: repetición de información, incapacidad de representar cierta información y posibles pérdidas de información al borrar tuplas. Para ello se estudian a continuación las técnicas formales de normalización.

## d) Concepto de Normalización y Necesidad de Normalizar.

Los modelos de datos son instrumentos, objetos y reglas, que nos ayudan a representar el Universo de Discurso (UD). El proceso de diseño de una base de datos consiste en representar un determinado UD mediante los objetos que proporciona el modelo de datos (estructuras) aplicando para ello las reglas de dicho modelo (restricciones inherentes).

Cuando realizamos un diseño en el modelo relacional, existen diferentes alternativas, pudiendo obtener diferentes esquemas relacionales, no todos ellos serán equivalentes y unos representarán mejor la información que otros.

En el modelo relacional el diseño puede realizarse de dos formas:

a) Obtener el modelo relacional directamente de la observación del UD.
b) Realizar el diseño del modelo E/R y transformarlo al modelo Relacional.

Las relaciones obtenidas pueden presentar los siguientes problemas:

- Incapacidad para representar ciertos hechos.
- Redundancia en la información, e incoherencias en la misma.
- Ambigüedades.
- Aparición en la base de datos de estados no validos en el mundo real, como son las anomalias en la modificación, inserción y borrado).

***Ejemplo.***

*ESCRIBE (autor, nacionalidad, cod_libro, titulo, editorial, año);*

*Presenta varios problemas:*

- Gran cantidad de redundancia. *La nacionalidad del autor se repite en cada ocurrencia del mismo. Cuando un libro tiene más de un autor la editorial y el año se repiten tambien.*

- Anomalías de modificación. *Puede ocurrir que se modifique el nombre de editorial en una fila sin modificarla en el resto que corresponden al mismo libro.*
- Anomalias de inserción. *No sería posible la inserción de un autor del que no hubiera ningun libro (cod_libro → Clave primaria), tampoco podría haber obras anonimas. La inserción de un libro con más de un autor obligaría a la repetición de tuplas.*
- Anomalias de borrado. *Si se quiere dar de baja un libro tambien se perdería información de los autores y viceversa.*

Ante cualquier duda de sí el modelo es correcto, es preferible la aplicación de un método formal de analisis → *Tería de la Normalización.*

Necesitamos formalizar una teoría en la que estudiemos las mejoras que da el estar en una forma normal o en otra y donde definamos formalmente las condiciones que debe cumplir una relación para estar en una forma o en otra. Esta teoría nos permitirá afrontar el diseño de una forma estricta y nos permitirá asegurar que nuestro diseño estará libre de una gran parte de problemas.

## • Formas Normales.

La teoría de normalización consiste en obtener esquemas relacionales que cumplan unas determinadas condiciones y se centra en las determinadas Formas Normales. Se dice que un esquema de relación está en una determinada forma normal si satisface un conjunto determinado de restricciones.

1FN, 2FN y 3FN → Codd.
FNBC → Boyce y Codd.
4FN y 5Fn → Fagin.

### a) Relación entre formas normales.

Una relación esta en 5FN sí tambien lo esta en todos las anteriores.
Una relación que esta en 1FN no tiene porque estar en 2FN.

Objetivo → Obtener la forma normal mayor posible.

### b) Dependencias.

Dependencias funcionales → 2FN, 3FN, FNBC
Dependencias multievaluadas → 4FN
Dependencias de proyección / combinación → 5FN

### c) Normalización: Las 4 primeras Formas Normales.

- **Primera Forma Normal (1FN).**
  Una relación se encuentra en primera forma normal cuando no hay grupos repetitivos en sus atributos. Todos los dominios de los atributos contienen únicamente valores atómicos.
  *Se eliminan las dependencias funcionales no totales.*

- **Segunda Forma Normal (2FN).**
  Una relación está en 2FN si además de estar en 1FN todos los atributos que no forman parte de ninguna clave candidata suministran información a cerca de la clave primaria.
  Toda relación cuya clave esta formada por un solo atributo está en 2FN.
  *Se eliminan las dependencias funcionales transitivas.*

***Ejemplo:***

*PRESTAMO (num_socio, nombre_socio, cod_libro, fecha_presentamo, editorial, pais)*

*Claves candidatas:*

*(num_socio, cod_libro)*
*(nombre_socio, cod_libro)*

*Para pasarlo a 2FN:*

*PRESTAMOS1 (num_socio, nombre_socio, cod_libro, fecha_presentamo)*

*Atributo fecha_prestamos, no forma parte de la clave pero suministra información de claves candidatas.*

*LIBROS (cod_libro, editorial, pais)*

- **Tercera Forma Normal (3FN).**
  Una relación está en 3FN si además de estar en 2FN, los atributos que no forman parte de ninguna clave candidata facilitan información sólo acerca de las claves y no acerca de otros atributos.
  Cada atributo no clave es dependiente no transitivamente de la clave primaria.
  *Se eliminan las dependencias funcionales para los cuales el determinante no es clave candidata.*

  *PRESTAMOS1 → 3FN*
  *LIBROS → pais facilita información a cerca de editorial (No 3FN).*

  *LIBROS1 (cod_libro, editorial)*
  *EDITORIALES (editorial, pais)*

- **Forma Normal de Boyce-Cood (FNBC).**
  Una relación estça en FNBC si lo está en 3FNy si además el conocimiento de las claves permite averiguar todas las relaciones existentes entre los datos de la relación. Las claves candidatas deben ser los únicos descriptores sobre los que se facilita información por cualquier otro atributo.
  Cada determinante debe ser una clave candidata, siendo un determiante, aquel atributo con el cual otro atributo tiene dependencia funcional total.
  *Se eliminan las dependencias multivaluadas que no sean funcionales.*

*En PRESTAMOS1 nombre_socio y num_socio se repiten por cada libro y prestamo. Nombre_socio se refiere a num_socio y viceversa. (Ninguno de ellos es clave aunque formen parte de ella).*

*Pasar a FNBC:*

*SOCIOS (num_socio, nombre_socio)*
*PRESTAMOS2 (num_socio, cod_libro, fecha_prestamo)*

Esquema :

LIBROS1 (cod_libro, editorial)
EDITORIALES (editorial, pais)
SOCIOS (num_socio, nombre_socio)
PRESTAMOS2 (num_socio, cod_libro, fecha_prestamo)

**d) Concepto de dependencia funcional.**

Las dependencias son propiedades inherentes al contenido semántico de los datos que se han de cumplir para cualquier extensión del esquema de relación y forman parte de las restricciones de usuario del modelo relacional. Las dependencias, muestran interrelaciones existentes entre los atributos del mundo real cuya semántica tratamos de incorporar a nuestra base de datos. Son invariantes en el tiempo.

Puesto que las dependencias contituyen una parte importante de la semantica de nuestro UD, deben de ser incluidas en los esquemas de relación. De modo que un esquema de relación será un par:

<A, DEP>

A → Conjunto de atributos de la relación.
DEP → Conjunto de dependencias existentes entre dichos atributos.

Las tres primeras formas normales tienen dependencias funcionales. En el proceso de normalización será fundamental indentificar todas las dependencias funcionales del universo del discurso cuyo diseño estamos realizando y procuraremos conservar dichas dependencias a lo largo de todo el proceso.

Dado el esquema de relación R(a, b), sean X, Y $\subseteq$ A. Se dice que Y depende funcionalmente de X, y se representa por $X \rightarrow Y \Leftrightarrow$ para cada valor de X existe un único valor de Y en todo momento. Si t1 y t2 son tuplas de R(A; B) la dependencia funcional que:

$$X \rightarrow Y \Leftrightarrow t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y]$$

# • Conclusión.

Por último incluiremos una pequña guía de cómo utilizar la teoría de la normalización dentro del diseño lógico. Supóngase que E/R es el esquema relacional obtenido de la transformación del diagrma Entidad/Relación. Estos puntos son una orientación de esta utilización:

a) Comprobar que todas las relaciones incluidas en E/R están en 1FN. Las únicas relaciones que podrían no estarlo son aquellas que provienen de objetos del diagrama Entidad/Relación (entidades o relaciones) que poseían algún atributo estructurado o multivaluado. Transoforma estas relaciones qa un conjunto de relaciones que estén en 1FN. Se denota a este nuevo esquema relacional como E/R (1FN).

b) Comprobar que todas las relaciones incluidas en E/R (1FN) están en 2FN. Para este punto no se consideraran las claves alternativas, esto es, se supondrá que todas las relaciones sólo poseen una clave candidata. De esta forma los problemas asociados a la definición de atributo no clave desaparecen, siendo más sencilla la aplicación de la definición 2FN. Si alguna relación no está en 2FN descomponerla en un conjunto de relaciones que estén en 3FN. Se denota a este nuevo esquema relacional E/R (2FN).

c) Comprobar que todas las relaciones incluidas en E/R (2FN) están en 3FN. Generlamente, debido a la transformación del diagrama Entidad/Relación al modelo Relacional, las relaciones del E/R (2FN) estarán en 3FN. Es posible que alguna relación no lo esté porque se incluya posteriormete alguna dependencia funcional entre un par de atributos no clave. Transofrmar estas relaciones a un conjunto de relaciones en 3FN.

d) Comprobar que todas las relaciones incluidas en E/R (3FN) están en FNBC. Para realizar esta tarea se han de considerar las claves candidatas. La presencia de claves candidatas y la posibilidad de solapamiento entre ellas es bastante remota, por lo que, generalmente, cada una de las relaciones del esquema relacional E/R (3FN) estará también en FNBC. De no ser así, aplicar la descomposición vista y obtener para la relación que no esté en FNBC un conjunto de relaciones que sí lo estén.

Como punto final de este apartado es interesante recordar que el objetivo final del diseño lógico, en cuanto al diseño de los aspectos estáticos, es obtener un esquema relacional en el cual cada una de las relaciones se encuentra en FNBC. Para ello, cada una de las relaciones incluidas en el esquema relacional obtenido aplicando las reglas de la transformación del diagrama Entidad/Relación será analizada para comprobar que cumple esta propiedad. De no ser así se descompondrá en las en las relaciones adecuadas. Por otra parte las posibles restricciones de integridad incluidas en el esquema conceptual deberán ser traducidas a expresiones equivalentes del cálculo relacional de tuplas o dominios utilizando como esquema relacioanl el obtenido después de la aplicación de la normalización.

# 6. SQL: El lenguaje de las Bases de Datos Relacionales.

- **Introducción al lenguaje SQL.**

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos informática. El nombre de "SQL" es una abreviación de Structured Query Language (Lenguaje Estructurado de Consultas). Como su nombre indica, SQL es un lenguaje informático que se utiliza para interactuar con un tipo de bases de datos, llamado base de datos relacional. SQL es un lenguaje de bases de datos relacionales, y ha llegado a ser popular junto con el modelo relacional de base de datos. La estructura tabular fila/columna de las bases de datos relacionales es intuitiva para los usuarios, manteniendo el lenguaje SQL sencillo y fácil de entender.

*SQL permite la definición de datos, la recuperación de datos, la manipulación de datos, y el control de acceso, así como la compartición de la información y la integridad de los datos. [GROFF98]*

*SQL es un lenguaje interactivo de consultas, es un lenguaje de programación de bases de datos, es un lenguaje de administración de bases de datos, es un lenguaje cliente/servidor, es un lenguaje de bases de datos distribuidas y es un lenguaje pasarela de bases de datos. [GROFF98]*

- **Características y ventajas de SQL.**

- Independencia de los proveedores.
- Portabilidad entre sistemas informáticos,
- Estándares de IBM (DB2).
- Acuerdos con Microsoft (ODBC).
- Fundamento relacional.
- Estructura de alto nivel parecida al inglés.
- Consultas ad hoc interactivas.
- Accesos a bases de datos desde programas.
- Múltiples vistas de los datos.
- Lenguaje completo de bases de datos.
- Definición dinámica de datos.
- Arquitectura cliente/servidor.

- **Los estándares ANSI/ISO.**

El trabajo sobre el estándar SQL oficial comenzó en 1982, cuando ANSI encargó a su comité X3H2 la definición de un lenguaje de base de datos relacional. El estándar ANSI para SQL resultante está basado en gran medida en el SQL de DB2, aunque contiene algunas diferencias importantes con respecto a él. Después de varias revisiones, el estándar fue oficialmente adoptado como estándar ANSI X3.135 en 1986, y como estándar ISO en 1087. Este estándar, revisado y ampliado ligeramente en 1989, es llamado generalmente "SQL-89" o estándar "SQL1".

Para afrontar pequeñas diferencias semánticas entre desarrolladores del estándar original, el comité ANSI continuó su trabajo proponiendo dos nuevos estándares, el SQL2 y el SQL3. En 1992 aprobó el estándar SQL2 llegando a ocupar casi 600 páginas mientras que el estándar orig1inal no llegaba a las 100 páginas. A pesar de su existencia, actualmente no hay ningún producto comercial disponible que implemente todas sus características.

Debido a las diferencias semánticas antes mencionadas y a que el estándar especifica ciertos detalles como "definidos por el fabricante" es posible ejecutar la consulta con dos implementaciones SQL diferentes, y producir dos conjuntos diferentes de resultados. Estas diferencias ocurren en el manejo de los valores nulos (valores NULL), las funciones de columnas y la eliminación de filas duplicadas.

- **Algunas diferencias entre SQL1 y SQL2.**

1. Relativas al < tipo de datos >.

   a) En el SQL del 89 se admitían los siguientes tipos de datos, bastante usuales en los productos actuales: INTEGER, SMALLINT, CHARACTER, DECIMAL, NUMERIC, REAL, FLOAT y DOUBLE PRECISION, que también se encuentran en varios lenguajes de programación.

   b) En el SQL2, además de los anteriores, se admiten los siguientes tipos de datos: CHARACTER VARYING, DATE, TIME, BIT, TIMESTAMP, INTERVAL y BIT VARYING. A este respecto cabe destacar que los tipos BIT y BIT VARYING contienen bits, que no son interpretados por el SGBD; en cuanto a INTERVAL, expresa la diferencia entre fechas u horas; por último, TIMESTAMP contiene año, mes, día, hora, minutos, segundos y, opcionalmente, fracciones de segundo, dando por tanto más facilidades para el tratamiento de la dimensión temporal.

2. Relativas a < expresión de valor >.

   Por expresión de valor se entiende cualquier expresión que dé como resultado un valor numérico, carácter, hora, fecha o intervalo. Puede ser tan simple como un dígito o tan complejo como una subconsulta.

3. Relativas a la < definición de restricción referencial >.

   a) Si <columnas y tabla referenciada> especifica una <lista de columnas de referencia>, ésta debe ser idéntica a una <lista de columnas únicas> en una <definición de restricción de unicidad> de la tabla referenciada. Por tanto, se puede establecer una restricción referencial con conjuntos de columnas únicas sin que sean necesariamente la clave primaria. En caso de no especificar la <lista de columnas de referencia>, entonces se entenderá que se referencia a la clave primaria.

   b) En caso de no especificar la <acción referencial disparada>, se considera que ésta es restringida.

   c) En realidad, las propuestas del estándar son más completas, debiéndose especificar otros aspectos de las restricciones de integridad referencial, como pueden ser el tipo de concordancia (esto es, si todos los valores de la columnas que referencian pueden ser nulos, no nulos o ambas cosas) o el modo de restricción (diferida, si se verifica al finalizar la transacción, o inmediata, si se efectúa al finalizar cada sentencia).

4. Relativas a la < operación de comparación >.

   Las operaciones de comparación son las usuales: mayor, menor, igual, mayor o igual, etc.

5. Relativas a la confidencialidad.

   La cláusula WITH GRANT OPTION, indica que el usuario que recibe un privilegio lo puede, a su vez, conceder a otros.

## • Falta de datos en SQL (Valores NULL).

Puesto que una base de datos es generalmente un modelo de una situación del mundo real, ciertos datos pueden inevitablemente faltar, ser desconocidos o no ser aplicables.

En la base de datos ejemplo, la columna CUOTA de la Tabla REPVENTAS contiene el objetivo de ventas de cada vendedor. Sin embargo, el vendedor más reciente aún no tiene asignada una cuota; este dato falta para esa fila de la tabla. Podríamos estar tentados de colocar un cero en la columna para este vendedor, pero esa no sería un reflejo preciso de la situación. El vendedor no tiene una cuota cero, la cuota simplemente es "desconocida" aún.

Análogamente, la columna DIRECTOR de la tabla REPVENTAS contiene el número de empleados de cada director de vendedores. Pero San Clark, el vicepresidente de ventas, no tiene director en la organización de ventas. Esta columna no es aplicable a Sam. De nuevo, se podría pensar en introducir un cero, o un 9999 en la columna, pero ninguno de estos casos sería el número de empleados realmente. Ningún valor es aplicable a esta fila.

SQL incorpora explícitamente los datos que faltan, son desconocidos o no son aplicables, a través del concepto de *valor nulo.*

*Un **valor nulo** es un indicador que dice a SQL (o al usuario) que el dato falta o no es aplicable. Por conveniencia, un dato que falta normalmente se dice que tiene el valor NULL, pero el valor NULL no es un valor de dato real. En vez de ello es una señal o un recordatorio de que el valor falta o es desconocido. [GROFF98]*

La tabla REPVENTAS muestra esta situación.

| Nombre | Edad | Oficina | Título | Contrato | Dir | Cuota | Ventas |
|---|---|---|---|---|---|---|---|
| Bill Adams | 37 | 15 | Rep Ventas | 02-12-1988 | 104 | 350.000 | 367.000 |
| Mary Jones | 31 | 11 | Rep Ventas | 10-12-1989 | 106 | 300.000 | 392.000 |
| Sue Smith | 48 | 21 | Rep Ventas | 12-10-1986 | 108 | 350.000 | 474.000 |
| Sam Clark | 31 | 11 | VP Ventas | 01-12-1988 | NULL | 275.000 | 299.000 |
| Tom Snyder | 41 | NULL | Rep Ventas | 01-02-1989 | 101 | NULL | 75.000 |

En muchas situaciones los valores NULL requieren una gestión especial por parte del SGBD. Por ejemplo, si el usuario solicita la suma de la columna CUOTA. ¿Cómo tendría que manejar el SGBD la falta de datos cuando calcule la suma? La respuesta viene dada por un conjunto de reglas especiales que gobiernan el manejo de los valores NULL en las diferentes sentencias y cláusulas SQL.

### • Consideraciones del valor NULL.

El comportamiento de los valores NULL en los test de comparación puede revelar que algunas nociones "obviamente ciertas" referentes a consultas SQL no son, de hecho, necesariamente ciertas. Por lo tanto es necesario considerar la gestión del valor Null cuando se especifica una condición de búsqueda. En la lógica trivaluada de SQL, una condición de búsqueda puede producir un resultado TRUE, FALSE o NULL. Sólo las filas en donde la condición de búsqueda genera un resultado TRUE se incluyen en los resultados de la consulta.

- ***Tratamiento de los VALORES NULOS en SQL2.***
  ***Diferencias con SQL1.***

## a) Operaciones aritméticas.

Cualquier operación aritmética sobre un campo nulo nos devolverá como resultado un valor nulo. Tomemos como ejemplo la siguiente tabla:

NULOS

| COL_A | COL_B |
|---|---|
| 15 | 10 |
| 35 | 35 |
| 140 | NULL |
| NULL | 100 |
| NULL | NULL |
| 7 | 110 |
| 33 | 60 |
| NULL | NULL |
| NULL | NULL |

Ejemplo de SUMA:

```
SELECT COL_A+COL_B
  FROM NULOS
```

Resultado:

| COL_A+COL_B |
|---|
| 25 |
| 70 |
| NULL |
| NULL |
| NULL |
| 117 |
| 93 |
| NULL |
| NULL |

b) **Funciones de columna y Valores NULL.**

SQL permite resumir datos de una base de datos mediante un conjunto de funciones de columna. Una función de columna SQL acepta una columna entera de datos como argumento y produce un único dato que resume la columna. El estándar ANSI/ISO especifica que los valores NULL de la columna sean ignorados y establece unas reglas precisas para gestionar los valores NULL en funciones columna.

- Si alguno de los valores de datos de una columna es Null, se ignora para el propósito de calcular el valor de las funciones de columna.
- Si todos los datos de una columna son Null, las funciones de columna SUM(), AVG(), MIN() y MAX() devuelven un valor Null; la función COUNT() devuelve un valor de cero.
- Si no hay datos en la columna (es decir, la columna está vacía), las funciones de columna SUM(), AVG(), MIN() y MAX() devuelven un valor cero.
- La función COUNT(*) cuenta filas, y no depende de la presencia o ausencia de valores Null en la columna. Si no hay filas, devuelve un valor de cero.

Ejemplos:
```
SELECT AVG(COL_A)
    FROM Nulos

SELECT SUM(COL_A)/COUNT(*)
   FROM Nulos
```

Resultado: AVG(COL_A)=46
SUM(COL_A)/COUNT(*)=25.5

**c) Comparaciones: Ejemplos.**

Dos valores nulos no son iguales ni son distintos, sino indeterminados.

```
SELECT *
   FROM NULOS
   WHERE COL_A=COL_B
```

Resultado:

| COL_A | COL_B |
|---|---|
| 35 | 35 |

```
SELECT *
   FROM NULOS
   WHERE COL_A<>COL_B
```

Resultado:

| COL_A | COL_B |
|---|---|
| 15 | 10 |
| 140 | NULL |
| NULL | 100 |
| 7 | 110 |
| 33 | 60 |

```
SELECT *
   FROM NULOS
   WHERE COL_A IS NULL
```

Resultado: Esta orden visualiza todas las filas en las que el campo perteneciente a la columna COL_A es nulo.

BETWEEN.

- Si la expresión de test produce un valor NULL, o si ambas expresiones definitorias del rango producen valores NULL, el test BETWEEN devuelve un resultado NULL.
- Si la expresión que define el extremo inferior del rango produce un valor NULL, el test BETWEEN devuelve FALSE si el valor de test es superior al límite superior, y NULL en caso contrario.
- Si la expresión que define el extremo superior del rango produce un valor NULL, es test BETWEEN devuelve FALSE si el valor del test es menor que el límite inferior, y NULL en caso contrario.

IN.

Si la expresión de test produce un valor Null, el test IN produce un valor Null. Todos los elementos en la lista de valores objetivo deben tener el mismo tipo de datos, y ese tipo debe ser comparable al tipo de dato de la expresión de test.

LIKE.

Se puede localizar cadenas que no se ajusten a un patrón utilizando el formato NOT LIKE del test de correspondencia de patrones. Es test LIKE debe aplicarse a una columna con un tipo de datos cadena. Si el valor del dato en la columna es Null, es test LIKE devuelve un resultado Null.

**d) Condiciones de búsqueda (AND, OR y NOT).**

Las condiciones de búsqueda simples, devuelven un valor TRUE, FALSE o NULL cuando se aplican a una fila de datos. Del mismo modo, los valores Null influyen en el resultado de las condiciones de búsqueda compuesta de una manera sutil. Ver las siguientes tablas.

Tabla de verdad de AND.

| AND | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | FALSE | NULL |
| FALSE | FALSE | FALSE | NULL |
| NULL | NULL | NULL | NULL |

Tabla de verdad de OR.

| OR | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | NULL |
| NULL | TRUE | NULL | NULL |

Tabla de verdad de NOT.

| NOT | TRUE | FALSE | NULL |
|---|---|---|---|
| | FALSE | TRUE | NULL |

*El estándar SQL2 añade otra condición lógica de búsqueda,* ***el test IS****, a la lógica proporcionada por AND, OR, y NOT,* ***que comprueba si el valor lógico de una expresión o comparación es TRUE, FALSE o UNKNOWN (NULL).***

**e) Ordenación de los resultados de una consulta (ORDER BY).**

Dependiendo del sistema gestor en uso los valores nulos serán los de mayor o los de menor peso. Tenemos los siguientes:

- DB/2 de IBM: NULL  Mayor peso. En ordenación ascendente serán los últimos.

*SELECT COL_A*
*FROM NULOS*
*ORDER BY COL_A*

Resultado:

| COL_A |
|---|
| 7 |
| 15 |
| 33 |
| 35 |
| 140 |
| NULL |
| NULL |
| NULL |
| NULL |

- SQL-SERVER : NULL  Menor peso. En ordenación ascendente serán los primeros.

*SELECT COL_A*
*FROM NULOS*
*ORDER BY COL_A*

Resultado:

| COL_A |
|---|
| NULL |
| NULL |
| NULL |
| NULL |
| 7 |
| 15 |
| 33 |
| 33 |
| 140 |

Se define para ordenar la salida de una consulta por los campos que se especifiquen a continuación. Sintaxis:

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

ORDER BY <especificación de columna><orden>{<especificación de columna><orden>}

<especificación de columna>=<nombre de columna>|<posición de columna>
<orden>=ASC|DESC

**Ejemplo:** Obtener el número de plazas libres que quedan para cada vuelo y ordenar el resultado de más a menos plazas libres. Para igual número de plazas ordénese por número de vuelo.

```
SELECT NUM_VUELO, SUM(PLAZAS-LIBRES)
   FROM RESERVAS
   GROUP BY NUM_VUELO
   ORDER BY 2 DESC, NUM_VUELO
```

El estándar SQL2 permite controlar la forma de ordenación que utiliza el SGBD para cada clave de ordenación. Esto puede ser importante cuando se trabaja con juegos de caracteres internacionales o para asegurar la portabilidad entre sistemas de juegos de caracteres ASCII y EBCDIC. Sin embargo, esta área de la especificación SQL2 es bastante compleja y la mayor parte de implementaciones de SQL ignoran temas de secuenciamiento de la ordenación o utilizan su propio esquema propietario para el control por parte del usuario de la secuencia de ordenación.

### f) UNION-UNION ALL.

Se define para recuperar, usando una única consulta, información que se obtiene a partir de más d una consulta. Sintaxis:

```
<SELECT>
UNION [ALL]
<SELECT>
        {UNION[ALL]
        <SELECT>}
```

Características:

Cada SELECT devuelve un conjunto de filas. La unión será la tabla resultado.
Condiciones de cada estructura SELECT:

- Todas deben ser iguales o compatibles una a una. Esto supone que por cada columna tengamos un único tipo de dato.
- Pueden ser completas (WHERE, GROUP BY, ...), exceptuando la cláusula ORDER BY, que se ubicará al final de la última SELECT.
- UNION sin ALL proporciona un resultado sin filas duplicadas. **Ejemplo:** Sacar una lista de todas aquellas ciudades para las que haya vuelo, ordenadas alfabéticamente.

```
SELECT ORIGEN
   FROM VUELOS
   UNION
        SELECT DESTINO
         FROM VUELOS
        ORDER BY 1
```

***Nota:***
***Las tablas VUELOS, RESERVAS, y AVIONES de la bases de datos de los ejemplos están en el ANEXO I.***

g) **DISTINCT.**

El estándar SQL2 permite que esta cláusula se aplicada a cualquier función de consulta, y permite que se utilicen expresiones como argumentos de las funciones así mismo. Además la cláusula DISTINT sólo puede ser utilizada una vez en una consulta. Pero si puede ser especificada una segunda vez dentro de una subconsulta. No elimina los valores nulos repetidos.

*SELECT DISTINCT COL_A*
*FROM NULOS*

Resultado:

| COL_A |
|---|
| 15 |
| 35 |
| 140 |
| NULL |
| NULL |
| 7 |
| 33 |
| NULL |
| NULL |

## h) Indices únicos: CREATE UNIQUE INDEX.

Una de las estructuras de almacenamiento físico proporcionadas por la mayoría de los sistemas de gestión de bases de datos basados en SQL es el índice. Un índice es una estructura que proporciona un acceso rápido a las filas de una tabla en base a los valores de una o mas columnas.

La ventaja de tener un índice es la de acelerar enormemente la ejecución de las sentencias SQL con condiciones de búsqueda que se refieren a las columnas indexadas. Una desventaja de tener un índice es que consume espacio en disco adicional. Otra desventaja es que el índice debe ser actualizado cada vez que se añade una fila a la tabla y cada vez que la columna indexada se actualiza en una fila existente. Esto impone recargos adicionales a las sentencias INSERT y UPDATE para la tabla..

*Sobre una columna de índice único sólo está permitida la existencia de un valor nulo.*

*CREATE UNIQUE INDEX IXNULOS*
*ON NULOS*
(COL_A)

Resultado: Devolvería un error, ya que existe más de un campo con NULL. Para este caso los nulos se interpretan como valores iguales.

### i) Valores Null en columnas de agrupación: GROUP BY.

El estándar SQL ANSI/ISO considera que dos valores Null son iguales a efectos de la cláusula GROUP BY. Si dos filas tienen Null en las mismas columnas de agrupación y valores idénticos en las columnas de agrupación no NULL, se agrupan dentro del mismo grupo de filas. Por lo tanto todos los nulos quedarán agrupados en el mismo grupo.

```
SELECT COL_A, COUNT(*)
   FROM NULOS
   GROUP BY COL_A
```

Resultado:

| COL_A | COUNT(*) |
|---|---|
| 15 | 1 |
| 35 | 1 |
| 140 | 1 |
| NULL | 4 |
| 7 | 1 |
| 33 | 1 |

## j) Condiciones de búsqueda de grupos: HAVING.

- Si la condición de búsqueda es TRUE, se retiene el grupo de filas y contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es FALSE, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es NULL, el grupo de filas se descarta u no contribuye con una fila resumen a los resultados de la consulta.

## k) Composiciones y los valores NULL en el estándar SQL2.

La operación de composición SQL combina información procedente de dos tablas mediante la formación de pares de filas relacionadas en las dos tablas. Los pares de filas que componen la tabla compuesta son aquéllos en los que las columnas asociadas en cada una de las dos tablas tienen el mismo valor. Si una de las filas de una tabla no se empareja en este proceso, la composición puede producir resultados inesperados, como muestran estas consultas teniendo en cuenta la tabla REPVENTAS:

**Ejemplo1:** Lista los vendedores y las oficinas en que trabajan.

```
SELECT Nombre, Oficina_Rep
   FROM Repventas
```

Resultado:

| Nombre | Oficina |
|---|---|
| Bill Adams | 15 |
| Mary Jones | 11 |
| Sue Smith | 21 |
| Sam Clark | 11 |
| Tom Snyder | NULL |

Ejemplo2: Lista los vendedores y las ciudades en que trabajan.

```
SELECT Nombre, Ciudad
   FROM Repventas, Oficinas
   WHERE  Oficina_Rep = Oficina
```

Resultado:

| Nombre | Ciudad |
|---|---|
| Bill Adams | Atlanta |
| Mary Jones | New York |
| Sue Smith | Los Angeles |
| Sam Clark | New York |

Aparentemente sería de esperar que estas dos consultas produjeran el mismo número de filas, pero la primera lista cinco vendedores y la segunda sólo cuatro. ¿Por qué? Porque Tom Snyder está actualmente sin asignar y tiene un valor Null en la columna Oficinas_Rep. Este valor Null no se corresponde con ninguno de los números de oficina en la tabla Oficinas. Como resultado desaparece de la composición.

La especificación SQL2 permite las composiciones externas por medio de la cláusula FROM, con una elaborada sintaxis que permite que el usuario especifique exactamente cómo se han de componer las tablas fuentes de una consulta. La cláusula FROM extendida también permite operaciones UNION entre tablas, y permite combinaciones complejas de composiciones y uniones.

- **Composiciones Internas en SQL2. Ejemplos:**

```
SELECT *
    FROM Chicas INNER JOIN Chicos ON Chicas.Ciudad = Chicos .Ciudad

SELECT *
    FROM Chicas INNER JOIN Chicos
    WHERE (Chicas.Ciudad = Chicos.Ciudad) AND (Chicas.Edad = Chicos.Edad)
```

La cláusula USING especifica una lista de nombres de columnas relacionadas separados por comas, que han de ser idénticos en ambas tablas.

Una composición entre dos tablas donde las columnas relacionadas son exactamente las columnas de las dos tablas que tienen nombres idénticos se denomina composición NATURAL.

```
SELECT *
    FROM Chicas NATURAL INNER JOIN Chicos
```

- **Composiciones Externas en SQL2. Ejemplos:**

```
SELECT *
    FROM Chicas FULL OUTER JOIN Chicos ON Chicas.Ciudad = Chicos.Ciudad

SELECT *
    FROM Chicas NATURAL FULL OUTER Chicos

SELECT *
    FROM Chicas FULL JOIN Chicos USING (Ciudad)

SELECT *
    FROM Chicas LEFT OUTER JOIN Chicos USING (Ciudad)
```

```
SELECT *
   FROM Chicas RIGHT OUTER JOIN Chicos USING (Ciudad)
```

- **Composiciones Cruzadas y Uniones en SQL2. Ejemplos:**

```
SELECT *
   FROM Chicas CROSS JOIN Chicos

SELECT *
   FROM Chicas UNION JOIN Chicos
```

- **Composiciones Multitabla en SQL2.**

Una de las principales ventajas de la cláusula FROM extendida del SQL2 es que proporciona un estándar uniforme para especificar composiciones internas y externas, productos y uniones. Otra ventaja, todavía más importante, es que permite una especificación muy clara de composiciones, productos y uniones de tres y cuatro tablas.

**Ejemplo1**: Lista todos los hijos con sus padres.

```
SELECT *
   FROM ((Chicas UNION JOIN Chicos) JOIN
            Padres ON (Padres.Hijo = Nombre))
```

**Ejemplo2:** Lista todos los hijos con sus padres incluyendo valore NULL.

```
SELECT *
   FROM ((Chicas UNION JOIN Chicos) LEFT OUTER JOIN
           Padres ON (Padres.Hijo = Nombre))
```

**Ejemplo3:** Lista Chicos y Chicas de las misma ciudad, incluyendo el nombre del padre del chico y el nombre de la madre de la chica en el resultado de la consulta.

```
SELECT *
   FROM ((Chicas LEFT JOIN Padres
                  ON ((Hijo = Chicas.Nombre) AND
                        Tipo = 'Madre')))
          JOIN
          ((Chicos LEFT JOIN Padres
                  ON ((Hijo = Chicos.Nombre) AND
                        Tipo = 'Padre')))
          USING (Ciudad)
```

**l) Subconsultas: SUBSELECT.**

Responde a la siguiente sintaxis:

*SELECT <lista_columnas>*
*FROM <lista_tablas>*
*WHERE <nombre_columna> <CONCATENADOR> (SELECT <nombre_columna>*
*FROM <lista_tablas>*
*WHERE <Predicado>)*

<CONCATENADOR>

Puede ser un operador de comparación o la cláusula IN
Operadores de comparación: >,<,>=,<=,=,<>

Restricciones: ha de exigirse que el resultado de la Subselect sea un único valor al usar como concatenador un operador de comparación. Si usamos IN puede devolver más de un valor.

Cada Select se ejecuta una única vez, desde la más interna, hasta la más externa.

**Ejemplo:** Se desea recuperar las plazas libres que hay en cada vuelo MADRID-LONDRES del día 20/02/92. {Las plazas libres es un campo de la tabla de reservas. En la tabla de vuelos tenemos el origen y el destino de cada vuelo.}

```
SELECT *
FROM RESERVAS
WHERE FECHA_SALIDA='20.02.1992'
AND NUM_VUELO IN(SELECT NUM_VUELO
                 FROM VUELOS
                 WHERE ORIGEN='MADRID'
                 AND DESTINO='LONDRES')
```

**m) ANY y ALL.**

Se usan para poder utilizar operadores de comparación con subselects que nos devuelvan más de un valor único como resultado.

- Si el test de comparación no es TRUE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ANY devuelve NULL. En esta situación, no se puede concluir si existe un valor producido por la subconsulta para el cual es test de comparación sea cierto; puede haberlo o no, dependiendo de los valores “correctos” de los datos Null (desconocidos).

- Si el test de comparación no es FALSE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ALL devuelve NULL. En esta situación, no se puede concluir si existe un valor producido por la subconsulta para el cual es test de comparación sea cierto; puede haberlo o no, dependiendo de los valores “correctos” de los datos Null (desconocidos).

**Ejemplo:**

*SELECT <lista_columna>*
*FROM <lista_tablas>*
*WHERE <nombre_columna> <CONCATENADOR> {ANY/ALL} (<Subselect>)*

Una expresión ANY es cierta si lo es para algún valor de los que devuelve la Subselect.
Una expresión ALL es cierta si lo es para todos los valores que devuelve la Subselect.

3>ANY(2,5,7) □ Cierto
3=ANY(2,5,7) □ Falso
3>ALL(2,5,7) □ Falso
3<ALL(9,10,11) □ Cierto

**Ejemplo:** Se quiere recuperar los aviones cuya longitud sea mayor que la envergadura de todos ellos.

*SELECT **
*FROM AVIONES*
*WHERE LONGITUD > ALL (SELECT ENVERGADURA*
*FROM AVIONES)*

ó también

*SELECT **
*FROM AVIONES*
*WHERE LONGITUD > (SELECT MAX(ENVERGADURA)*
*FROM AVIONES)*

ANY e IN tienen la misma función.
3 = ANY(2,3,5) y 3 IN (2,3,5) ----devuelven ambos Cierto.

## n) Subselects correlacionadas.

Son un tipo especial de subselect. La sintaxis es similar:

*SELECT <lista_columnas>*
*FROM <nombre_tabla_externa>*
*WHERE <nombre_columna> <CONCATENADOR> (SELECT <nombre_columna>*
*FROM <nombre_tabla_interna>*
*WHERE <Predicado>)*

En <Predicado> habrá una sentencia del tipo
*<nombre_columna_tabla_interna> <operador> <nombre_columna_tabla_externa>*

Las formas de ejecutar una subselect ordinaria y una correlacionadas son diferentes. Las subselects correlacionadas obedecen al siguiente algoritmo:

**ALGORITMO Subselect_Correlacionada**

1. Seleccionar fila de tabla externa
2. Ejecutar SELECT interno
3. Evaluar la condición del WHERE externo
   - Cierto: la fila seleccionada en 1 será una fila de salida
4. Si existe alguna fila más en la tabla externa ir al paso 1

**Ejemplo:** Se desea recuperar las reservas cuyo número de plazas libres sea mayor que la media para ese mismo vuelo.

```
SELECT *
  FROM RESERVAS, A
  WHERE PLAZAS_LIBRES > (SELECT AVG(PLAZAS_LIBRES)
                         FROM RESERVAS
                         WHERE NUM_VUELO=A.NUM_VUELO)
```

RESERVAS

| NUM_VUELO | FECHA_SALIDA | PLAZAS_LIBRES |
|---|---|---|
| IB740 | 20.02.92 | 5 |
| IB740 | 25.02.92 | 15 |
| IB740 | 03.03.92 | 10 |

Resultado: AVG(PLAZAS_LIBRES) = 10

## o) Alias de tablas.

Es un sobrenombre que se le da a una tabla y que debe ser único para toda la consulta. Se escribe dejando un blanco detrás del nombre de la tabla a la cual se va a calificar. Los alias de tablas se requieren en consultas que afectan a autocomposiciones. Sin embargo se pueden utilizar un alias en cualquier consulta.

La única exigencia para marcar las tablas en la cláusula FROM es que todas las marcas de una cláusula FORM determinada deben ser distintas unas de otras. La especificación SQL2 permite opcionalmente la clave AS entre un nombre de tabla y un nombre de alias.

**Ejemplo:** Se quiere recuperar los aviones que tienen menos de 1 hora y cuarto de recorrido como término medio.

VUELOS

| NUM_VUELO | ORIGEN | DESTINO | DISTANCIA |
|---|---|---|---|
| B747 | MADRID | LONDRES | 10000 |
| B747 | MADRID | PARIS | 4000 |

AVIONES

| NUM_VUELO | VELO_CRUC |
|---|---|
| B747 | 350 |
| B749 | 325 |
| B754 | 400 |

v=e/t, t>e/v, 1.25>e/v, v*1.25 > AVG(DISTANCIA)

```
SELECT *
FROM AVIONES
WHERE 1.25*VELO_CRUC > (SELECT AVG(DISTANCIA)
                        FROM VUELOS
                        WHERE NUM_VUELO=AVIONES.NUM_VUELO)
```

**p) EXISTS-NOT EXISTS.**

Se define para comprobar la existencia o ausencia del valor devuelto por una Subselect. Una expresión con EXIST devuelve Cierto si la Subselect nos devuelve al menos un valor.

WHERE EXISTS (<Subselect>)  Cierto

**Ejemplo:** Seleccionar toda la información de vuelos para aquellos que tengan origen Madrid y en los que queden plazas libres.

```
SELECT *
FROM VUELOS
WHERE ORIGEN='MADRID'
AND EXISTS (SELECT *
            FROM RESERVAS
            WHERE PLAZAS_LIBRES > 0
            AND NUM_VUELO=VUELOS.NUM_VUELO)
```

## q) Vistas en SQL2.

Las tablas de una base de datos definen la estructura y organización de sus datos. Sin embargo, SQL, también permite mirar los datos almacenados de otros modos mediante la definición de vistas alternativas de los datos. Una vista es una consulta SQL que está permanentemente almacenada en la base de datos y a la que se le asigna un nombre. Los resultados de una consulta almacenada son “visibles” a través de la vista y SQL permite acceder a estos resultados como si fueran, de hecho, una tabla “real” en la base de datos. Las vistas son parte importante de SQL por varias razones:

- Las vistas permiten acomodar el aspecto de una base de datos de modo que diferentes usuarios la vean desde diferentes perspectivas.
- Las vistas permiten restringir acceso a los datos, permitiendo que diferentes usuarios sólo vean ciertas filas o ciertas columnas de una tabla.
- Las vistas simplifican el acceso a la base de datos mediante la representación de la estructura de los datos almacenados del modo que sea más natural a cada usuario.

*Una vista es una “tabla virtual” en la base de datos cuyos contenidos están definidos por una consulta. Para el usuario de la base de datos, la vista parece igual que una tabla real, con un conjunto de columnas designadas y filas de datos. Pero a diferencia de una tabla real, una vista no existe en la base de datos como conjunto almacenado de valores [GROFF98].*

Las reglas especificas que determinan si una vista puede ser actualizada o no varían de un producto SGBD a otro, y suelen estar bastante detalladas.

El estándar SQL2 incluye una amplia definición de vistas actualizables junto a una cierta libertad para que pueda existir variación entre los distintos productos SGBD.

De igual forma el estándar SQL2 formalizó el soporte para la eliminación de vistas a través de la sentencia DROP VIEW. También tiene en cuenta lo que ocurre cuando un usuario intenta eliminar una vista y la definición de otra vista depende de ella, obligando a especificar una de las dos opciones (CASCADE o RESTRICT).

**Ejemplo:** Supongamos que se han creado dos vistas sobre la Tabla REPVENTAS con las siguientes sentencias CREATE VIEW:

```
CREATE VIEW Repeste AS
    SELECT *
        FROM Repvntas
        WHERE Oficina_Rep IN (11, 12, 13)

CREATE VIEW Repny AS
    SELECT *
        FROM Repeste
        WHERE Oficina_Rep = 11
```

En el estándar SQL2 la siguiente sentencia eliminará las dos vistas de la base de datos:

```
DROP VIEW Repeste CASCADE
```

Mientras que la siguiente sentencia fallará devolviendo un error, ya que la opción RESTRICT le indica al SGBD que elimine la vista únicamente si no existen vistas que dependan de ella. Esto proporciona una precaución adicional contra los efectos laterales no deseados de la sentencia DROP VIEW.

```
DROP VIEW Repeste RESTRICT
```

- **Catálogo del sistema o Diccionario de Datos.**

Un sistema de gestión de bases de datos debe mantener gran cantidad de información referente a la estructura de la base de datos con el fin de efectuar sus funciones de gestión de datos. En una base de datos relacional esta información está almacenada típicamente en el catálogo de sistema. Es el alma de un sistema gestor. Se define como un conjunto de tablas que forman una base de datos, y son definidas y mantenidas automáticamente por el sistema gestor. La información del catálogo del sistema describe las tablas, las vistas, las columnas, los privilegios y otras características estructurales de la base de datos.

Las tablas del catálogo de sistema se crean automáticamente al crear la base de datos. El SGBD se refiere constantemente a los datos del catálogo de sistema cuando procesa las sentencias SQL. Un ejemplo de realiza esta tarea es el siguiente:

```
SELECT *
  FROM VUELOS
```

1- El sistema busca en el catálogo si existe la tabla VUELOS.
2- Verifica si el usuario tiene acceso a esa información.
3- Se pregunta cuáles y cuántas columnas tiene la tabla VUELOS.

**DB2 IB**
**SYSTABLES:** una fila por cada tabla definida en la instalación.
**SYSCOLUMNS:** una fila por cada columna definida.
**SYSINDEXES:** una fila por cada índice definido.
**SYSVIEW:** una fila por cada vista.
**SYSTABAUTH:** una fila por cada autorización definida.

Todas las tablas son directamente consultadas por usuarios autorizados.
**ADM (Administrador):** es la persona que concede autorizaciones a los usuarios.
Un usuario autorizado puede efectuar operaciones del tipo:

```
SELECT *
   FROM SYSTABLES
   WHERE NAME='RESERVAS'
```

| NAME | DBNAME | CARD |
|---|---|---|
| RESERVAS | AEROPUERTO | 10 |

**DBNAME:** Nombre de la BD a la que pertenece la tabla.
**CARD:** Número de filas de la tabla.
**OWNER:** Usuario creador de la tabla.

```
SELECT *
  FROM SYSCOLUMNS
  WHERE DBNAME='RESERVAS'
```

Resultado: Nos da la información sobre todas las columnas que pertenecen a la tabla reservas.

| NAME | TBNAME | COL_NO | COL_TYPE | LENGTH | NULLS |
|---|---|---|---|---|---|
| NUM_VUELO | RESERVAS | 1 | CHAR | 6 | N |
| FECHA_SALIDA | RESERVAS | 2 | DATE | 8 | N |

**TBNAME:** Nombre de la tabla.
**COL_NO:** Posición de la columna en la tabla.
**COL_TYPE:** Tipo de dato
**LENGTH:** Longitud del dato de la columna.
**NULLS:** Indica si se permite valor nulo.

Debido a la creciente importancia de las herramientas de base da datos de propósito general que deben acceder al catálogo de sistema, es estándar SQL2 propuesto incluye una especificación de un conjunto de vistas que proporciona un acceso estandarizado a la información que normalmente se encuentra en el catálogo de sistema. Un SGBD que cumpla el estándar SQL2 debe soportar estas vistas, que en su conjunto se denominan INFORMATION_SCHEMA (esquema de información).

- # Integridad de datos en SQL2.

Para preservar la consistencia de los datos almacenados, un SGBD Relacional impone típicamente una o más restricciones de integridad de datos. Estas restricciones restringen los valores que pueden ser insertados en la base de datos o creados mediante una actualización de la base de datos. Varios tipos diferentes de restricciones de integridad de datos suelen encontrarse en las bases de datos relacionales, incluyendo:

- **Datos requeridos:** No se permite la ausencia de valor o que contengan valores Null.

- **Chequeo de validez**, mediante restricciones de comprobación.

- **Integridad de entidad.** *La clave primaria* de una tabla debe tener un único valor para cada fila e la tabla o si no la base de datos perderá su integridad como modelo del mundo exterior. Los valores NULL presentan un problema cuando aparecen en la clave primaria de una tabla o en una columna que está especificada en una restricción de unicidad. Supongamos que se intentase insertar una fila con una clave primaria que fuera NULL (o parcialmente NULL, si la clave primaria está compuesta por más de una columna). Debido al valor NULL, el SGBD no puede decidir concluyentemente si la clave primaria está o no duplicada con respecto a otra ya existente en la tabla. La respuesta debe ser "quizá", dependiendo del valor "real" del dato que falta (NULL).
  Por esta razón, SQL requiere que toda columna que forma parte de una clave primaria y toda columna designada en una restricción de unicidad deben ser declaradas NOT NULL.

- **Integridad referencial.** A diferencia de las claves primarias, *las claves ajenas* de una base de datos relacional se permite que contengan valor NULL. El estándar SQL2 trata este problema proporcionando al administrador de la base de datos más control sobre la administración de los valores NULL en las claves ajenas para las restricciones de integridad. La restricción de integridad en la sentencia CREATE TABLE permite dos opciones:
  a) La opción MATCT FULL requiere que las claves ajenas de la tabla hijo coincidan totalmente con una clave primaria de la tabla padre. Con esta opción, ninguna parte de la clave ajena puede contener el valor NULL, con lo que el problema de la gestión del valor NULL en las reglas de supresión y actualización no se produce.
  b) La opción MATCH PARTIAL permite valores NULL en artes de una clave ajena, mientras que los valores que no son NULL coinciden con las partes correspondientes de alguna clave primaria de la tabla padre. Con esta opción, la gestión del valor NULL en las reglas de supresión y actualización prosigue como se describió anteriormente.

- **Reglas comerciales.** Las reglas comerciales pueden ser forzadas por SGBD mediante el mecanismo disparador popularizado por Sybase y SQL Server. Los disparadores permiten al SGBD tomar acciones complejas en respuesta a eventos tales como sentencias INSERT, DELETE o UPDATE. Las restricciones de comprobación proporcionan un mecanismo más limitado para incluir las reglas comerciales en la definición de una base de datos y de hacer que el SGBD las fuerce.

- **Consistencia de los datos.**

# Anexo I: Ejemplos del Lenguaje SQL.

BASE DE DATOS: ***AEROPUERTO***

VUELOS

| NUM_VUELO | ORIGEN | DESTINO | HORA_SALIDA | TIPO_AVION |
|---|---|---|---|---|
| IB600 | MADRID | LONDRES | 10:30:00 | 320 |
| BA467 | MADRID | LONDRES | 20:40:00 | 73S |
| IBO640 | MADRID | BARCELONA | 06:45:00 | 320 |
| IB3742 | MADRID | BARCELONA | 09:15:00 | 72S |

RESERVAS

| NUM_VUELO | FECHA_SALIDA | PLAZAS_LIBRES |
|---|---|---|
| IB600 | 20/02/1992 | 46 |
| IB600 | 21/02/1992 | 80 |
| IB600 | 22/02/1992 | 91 |
| BA467 | 20/02/1992 | 32 |

AVIONES

| TIPO | CAPACIDAD | LONGITUD | ENVERGADURA | VELOCIDAD_CRUCERO |
|---|---|---|---|---|
| D9S | 110 | 38.30 | 28.50 | 815.0 |
| 320 | 187 | 42.15 | 32.60 | 853.0 |
| 72S | 160 | 36.20 | 25.20 | 820.0 |
| 73S | 185 | 44.10 | 30.35 | 815.0 |

**Ejemplo1:** Obténgase la última hora de salida para cada destino de los vuelos realizados por aviones capaces de almacenar más combustible que un tercio de la media que pueden almacenar los demás aviones.

```
SELECT DESTINO, MAX(HORA_SALIDA)
FROM VUELOS
WHERE TIPO_AVION IN (SELECT TIPO
                     FROM A
                     WHERE COMBUSTIBLE>1/3*(SELECT AVG(COMBUSTIBLE)
                                            FROM AVIONES
                                            WHERE TIPO<>A.TIPO)
```

**Ejemplo2:** Crear una vista sobre la tabla vuelos con las columnas ORIGEN y DESTINO para aquellos vuelos que no sean de IBERIA. Visualizar el contenido de la lista para los vuelos que no partan de Madrid. Borrar la vista.

```
CREATE VIEW V_VUELOS (V_ORIGEN, V_DESTINO)
AS SELECT ORIGEN, DESTINO
      FROM VUELOS
      WHERE NUM_VUELO NOT LIKE 'IB%'

SELECT *
   FROM V_VUELOS
   WHERE V_ORIGEN<>'MADRID'

DROP VIEW V_VUELOS
```

**Ejemplo3:** Visualice los tipos de avión, el doble de su longitud y la mitad de su envergadura, para aquellos aviones con envergadura mayor que la media y que realizan vuelos desde o hacia Barcelona, ordenándolos de mayor a menor longitud.

```
SELECT TIPO_AVION, 2*LONGITUD, .5*ENVERGADURA
   FROM AVIONES, VUELOS
   WHERE ENVERGADURA>(SELECT AVG(ENVERGADURA)
                      FROM AVIONES) AND
                      AVIONES.TIPO_AVION=VUELOS.TIPO_AVION AND
                      (ORIGEN='BARCELONA' OR DESTINO='BARCELONA)
                      ORDER BY 2 DESC
```

**Ejemplo4:** Visualice las tres primeras letras de los orígenes y destinos de los vuelos realizados por aviones con longitud mayor con longitud mayor que la media y envergadura menor que 2/3 de la máxima envergadura, ordenados alfabéticamente por destino.

```
SUBSTRING (SQL)
(SUBSTRNG), (DB2)

SUBSTRING (string, posición, nºcaracteres)    nom_col / cadena con comillas (")

SELECT SUBSTRING (ORIGEN, 1, 3), SUBSTRING (DESTINO, 1, 3)
FROM VUELOS
WHERE TIPO_AVION IN (SELECT TIPO
                     FROM AVIONES
                     WHERE LONGITUD > (SELECT AVG(LONGITUD)
                                       FROM AVIONES) AND ENVERGADURA*3/2
                                          < (SELECT MAX(ENVERGADURA)
                                             FROM AVIONES)
                                             ORDER BY 2
```

**Ejemplo5:** Visualice el total de plazas libres por número de vuelo para aquellos realizados desde Madrid a Barcelona o Sevilla y que recorran una distancia mayor que la media de todos los vuelos que salen de Madrid, ordenándolos de menor a mayor.

```
SELECT SUM(PLAZAS_LIBRES), NUM_VUELO
  FROM RESERVAS, VUELOS
  WHERE RESERVAS, NUM_VUELO=VUELOS.NUM_VUELO
  AND ORIGEN='MADRID'
  AND DESTINO IN ('BARCELONA', 'SEVILLA')
  AND DISTANCIA > (SELECT AVG(DISTANCIA)
                   FROM VUELOS
                   WHERE ORIGEN='MADRID')
                   ORDER BY 1
```

**Ejemplo6:** Obtener para cada número de vuelo el total de plazas libres de los vuelos que recorran distancias menores que 2/3 de la media de las distancias recorridas por vuelos de otras compañías.

```
SELECT NUM_VUELOS, SUM(PLAZAS_LIBRES)
   FROM RESERVAS, VUELOS
   WHERE RESERVAS.NUM_VUELO=VUELOS.NUM_VUELOS
   AND DISTANCIA*3/2 < (SELECT AVG(DISTANCIA)
                        FROM VUELOS
                        WHERE SUBSTRING (NUM_VUELO, 1, 2) <>
                              SUBSTRING (VUELOS.NUM_VUELO, 1, 2)
```

# *Anexo II: Bibliografía.*

[DEMI93] Adoración de Miguel; Mario G. Piattini
Concepción y Diseño de Bases de Datos
Rama, España 1993.

[DATE93] C. J. Date
Introducción a los Sistemas de Bases de Datos. Vol I (5ª edic.)
Addison-Wesley Iberoamericana, 1993.

[BATI94] C. Batini; S. Ceri; S. B. Navathe
Diseño Conceptual de Bases de Datos. Un enfoque de entidades-interrelaciones
Addison-Wesley/Diaz de Santos 1994.

[RIVE88] E. Rivero Cornelio
Bases de Datos Relacionales
Paraninfo, España 1988.

[LUCA93] Angel Lucas Gómez
Diseño y Gestión de Bases de Datos
Paraninfo, España 1993.

[MOTA94] Laura Mota Herranz; Matilde Celma Giménez; J. C. Casamayor Ródenas
Bases de Datos Relacionales: Teoría y Diseño
Universidad Politécnica Valencia, España 1994.

[ELMA97] R. Elmasri; S. B. Navathe
Sistemas de Bases de Datos: Conceptos fundamentales
Addison-Wesley Iberoamericana, 1997.

[GROFF98] James R. Groff; Paul N. Weinberg
Lan Times, Guía de SQL
Osborne McGraw-Hill, 1998

[ABBEY98] Michael Abbey; Michael J. Corey
Oracle8: Guía de aprendizaje
McGraw-Hill, 1998

[APUNTES] Francisco Ruiz González; Manuel Ortega Cantero
Apuntes de la asignatura Bases de Datos.
Escuela Superior de Informática de Ciudad Real
Universidad de Castilla la Mancha, 1998/99